VLSI Structures for Viterbi Receivers: Part I—General Theory and Applications

P. GLENN GULAK AND EDWARD SHWEDYK, SENIOR MEMBER, IEEE

Abstract — A taxonomy of VLSI grid model layouts is presented for the implementation of certain types of digital communication receivers based on the Viterbi algorithm. We deal principally with networks of many simple processors connected to perform the Viterbi algorithm in a highly parallel way. Two interconnection patterns of interest are the "shuffle-exchange" and the "cube-connected cycles." The results are generally applicable to the development of area-efficient VLSI circuits for decoding convolutional codes, coded modulation with multilevel/phase signals, punctured convolutional codes, correlatively encoded MSK signals and for maximum likelihood sequence estimation of $M$-ary signals on intersymbol interference channels. In a companion paper, we elaborate on how the concepts presented here can be applied to the problem of building encoded MSK Viterbi receivers. Lower bounds are established on the product (chip area)/(baud rate)$^2$ and on the energy consumption that any VLSI implementation of the Viterbi algorithm must obey, regardless of the architecture employed or the intended application.

I. INTRODUCTION

The proliferation of digital information sources and sinks in our society has brought with it a greater need to convey this commodity accurately, rapidly, and inexpensively under the constraints of finite bandwidth and finite power. 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 Von-Neumann type 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, with one million transistor VLSI circuits on a single chip being promised shortly.

Complementing this activity, there have been persistent efforts in the literature directed towards developing VLSI structures for solving various types of dynamic programming problems in computer science [1]–[4]. This is a noteworthy development in that many important channel and source decoding tasks in digital communications can be solved by a dynamic programming approach using the algorithm developed by Viterbi [5]. Hence, it is natural to ask what types of parallel processing structures are appropriate in VLSI circuit technology for realizing digital communication receivers based on the Viterbi algorithm (VA).

In response to this query, this paper identifies a taxonomy of VLSI architectures (which contain perhaps hundreds or thousands of simple interacting processors) for certain types of digital communication detectors and decoders. Characteristic of these processor organizations, in a VLSI medium, is that the wire area dominates the processor area.

The presentation of this paper is organized into five sections. Initially, we discuss the Viterbi algorithm as a way of introducing the notation and terminology. Next, an abstraction called the VLSI grid model is introduced, which allows us to conveniently describe the processing architectures for the Viterbi algorithm. Sections III and IV present the taxonomy of VLSI architectures for the VA which are coarsely subdivided in each section, respectively, as having either small interprocessor wire area or large interprocessor wire area. Finally, in the fifth section we look at the tradeoff between the baud rate and VLSI chip area required that any VLSI implementation of the Viterbi algorithm must obey. In a companion paper, the design details of encoded MSK Viterbi receivers are presented, as a specific application of the concepts presented here.

II. THE VITERBI ALGORITHM

The Viterbi algorithm was originally invented to detect convolutionally encoded data symbols [5]. About twelve years ago, Forney suggested the application of the Viterbi...
algorithm to symbol-by-symbol data transmission [6]. The Viterbi algorithm can be thought of as a dynamic programming solution to the problem of estimating the state sequence of a finite-state Markov process observed in memoryless noise [7]. The basic theory behind the Viterbi decoding algorithm is widely available in the literature [8]; a good survey is given by Forney [9]. This section repeats some of the basic concepts in simplified form as a convenient way of introducing relevant terminology.

The block diagram of a typical digital communication system [10] is illustrated in Fig. 1. Our objective is to provide efficient VLSI architectures for source and channel decoding operations in the receiver, when these are based on the Viterbi algorithm.

Characteristic of the VA is the concept of a trellis, which is pictorially illustrated in Fig. 2(a). The trellis diagram is a graphical representation of a state diagram drawn as a function of discrete time. Each discrete time step is a single baud interval $T$, and corresponds to one stage of the trellis. The number of stages in the trellis diagram corresponds to the length of the input data sequence. The total number of nodes or states at each time step is $m^v$, where $m$ is the alphabet size and $v$ is a parameter known as the algorithm memory length. One state exists for each of the possible messages that contain the last $v$ symbols of the transmitted sequence. Fig. 2(a) illustrates the trellis diagram for an algorithm memory of two and a binary alphabet. Possible state transitions are illustrated by edges in the trellis diagram.

Two entries are stored at each node or state. One entry is the path metric, which indicates the likelihood of the hypothesis for which that state could exist. Path metrics, throughout this paper, are referred to as state metrics and are identified with individual processors. The other entry, called a survivor sequence, represents the earlier symbols, which led to the presumed present state, through valid state transitions. The survivor sequence eventually provides estimates of the transmitted source symbols.

Once each baud interval, state metrics are updated by adding a measure of unlikelihood, called a branch metric, defined by valid state transitions from each possible previous state into a given present state. The lowest resulting sum defines the new path metric (and trajectory) for this 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.

In a practical implementation of the VA, many pragmatic issues must be accommodated such as truncation of the survivor sequence, extraction of the estimated source symbols from the survivor sequence, and state metric overflow control in finite word-length registers. A truly parallel, though impractical, implementation of this algorithm could be realized by providing both: 1) a single processor for each node or state in the complete trellis diagram, and 2) wires along each state transition to convey state metrics and survivor sequences. However, the physical resources required would be proportional to the number stages in the trellis (i.e., to the length of the input data sequence). Note, however, that each stage of the trellis diagram has constant geometry. This fact can be exploited in creating an equivalent computational structure by taking a single stage of a trellis diagram and inserting an edge between equivalent initial and final states, for each of the $m^v$ states. These additional edges, known as recirculation edges, each transport a state metric and survivor sequence, every time step, from the final (back) to the initial (front) equivalent state. Fig. 2(b) illustrates the concept of a trellis
with recirculation edges, which is in fact a specialized class of bipartite graphs. Each of the concurrent architectures for realizing the VA, discussed in this paper, exploits this concept in a unique manner so that the computational demands of the algorithm can be structured.

The VA is not demanding because its algorithm is complex in the conceptual sense. Rather, the essence of the algorithm is a relatively simple procedure of add, compare, and select operations that must be applied to a huge number of basic "nodes" or "states." Unfortunately, the number of nodes grows as $m^v$, where $m$ is the alphabet size and $v$ is 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, until this time, is the type of topologies that are appropriate, and how "expensive" (with respect to an area criterion) is the wiring, for embedding these transistors.

Fig. 3 illustrates a parallel hardware implementation of the VA for an algorithm memory of two and a binary alphabet, using the concept of a single-stage trellis with recirculation edges. The design consists essentially of four Add-Compare-Select processing elements (one for each state) along with a substantial amount of interprocessor wiring. Due to the nature of the trellis, the organized placement and routing of the processing elements becomes an increasingly difficult (and area intensive) task as the algorithm memory length and/or alphabet size increases. In Section V that follows, two orderly schemes for placing the processors and their associated wiring are presented.

Historically, hardware implementations of the VA [11]–[15] have, in general, dealt with discrete integrated circuits and microprocessor realizations, where memory and processing elements themselves were considered to be the dominant cost factors and not the interprocessor wiring and its associated propagation delay and area requirements. A recent VLSI implementation of the VA, containing 16 simple processors that operate concurrently on a single die, for convolutional decoding [16] exploited the inherent parallelism of the algorithm to achieve a 37 Mbit/s throughput rate. However, it was found that 38 percent of the area on the chip was occupied by the wires that carry control and data to the processing elements. This provides an existence proof that the technology is mature enough to realize small algorithm memory versions of the VA directly; however, until this time, a general design methodology has yet to be identified to guide future developments.

The space and time complexity of the algorithm is well known. With an algorithm memory of $v$ and alphabet size of $m$, the algorithm requires $2m^v$ storage locations, one for each of the state metrics $\Gamma(a)$ and one each for the truncated survivor listing $\hat{z}(a)$ of $\delta$ symbols (each symbol is one of $m$). Computationally, in each unit of time the algorithm must make $m^{v+1}$ additions, one for each transition, and $m^v$ comparisons among the $m^{v+1}$ results. Thus, the amount of storage is proportional to the number of
states and the amount of computation to the number of transitions. However, the notion of space or memory requirements gives no consideration to the VLSI area required to transmit or route these results to processing elements. Associated with this is an energy cost of communicating information throughout the system not considered in the classical space-time model. In order to move away from the abstract notion of "space" to a more realistic accounting of "area" we are forced to resort to a computational model for VLSI known as the VLSI grid model.

### III. THE VLSI GRID MODEL

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 versus expensive devices). The expense in area of transmitting a result from where it is produced to where it is needed next can often exceed the cost in area of the transistor(s) producing the result in the first place. In order to account for the area devoted to processors and to interprocessor wiring, we resort to a construct known as the VLSI grid model.

In the VLSI grid model, first proposed by Thompson [17], a VLSI chip can be viewed as a computation graph whose vertices are called nodes and whose edges are called wires. Nodes are processors constructed of transistors and are responsible for the computation of Boolean functions. Wires are simply electrical connections and are responsible for both the transfer of information between nodes and the distribution of the power supply and timing information to nodes. The salient features of the VLSI grid model are: 1) Processing logic or storage at a node requires constant area. 2) Wires are laid out on a rectangular grid of unit squares. Thus, wires can cross only at right angles and computation graphs have vertex degrees less than or equal to four. 3) The chip is synchronous (propagation delay is independent of wire length); 4) semilective and semilocal (inputs of the problem are available at only one time and one place during the symbol interval); 5) when-determinate and where-determinate (inputs and outputs appear at pre-specified times and ports), and 6) word-local (all the bits of a given word enter or exit same port). The area $A$ of the circuit is defined as the area, in unit squares, of the smallest rectangle enclosing the circuit. In what follows, VLSI grid model layouts of various architectures capable of solving the VA are presented.

### IV. LAYOUTS WITH SMALL WIRE AREA

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 for completeness. The implementation suffers from being processor poor and from being input/output (I/O) bound. The uniprocessor must coordinate $O(m^n)$ random accesses to the processor's I/O memory and perform $O(m^{n+1})$ arithmetic operations each symbol interval $T$. Consequently, the hardware logic speed must be $\Omega(m^{n+1}/T)$ operations per second. 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, in principle, the smallest area solution to the VA imaginable. Adding more processing elements has the potential of increasing throughput, as demonstrated in the discussion which follows.

### V. LAYOUTS WITH LARGE WIRE AREA

In this section we consider layouts for the VA dominated by large interprocessor wire area. Two schemes based on the shuffle-exchange (SE) and cube-connected cycles (CCC) are identified for the placement and routing of the required ACS processing elements. Both of these interconnection schemes are important in that recently established theoretical results have provided constructive proofs of how to layout these networks in a minimum amount of area. These structures are generalized to accommodate arbitrary alphabet sizes. They can support the decoding of high-speed data streams since they have the right topology to move operands from where they are produced to where they are needed next in one routing step.

Shuffle-exchange graphs were originally proposed in 1971 as an interconnection methodology for parallel computation [19]. With the advent of VLSI, the question of how to best lay out the SE graph on a grid using as little area as possible has been extensively investigated [20], [21]. Presently, we will show that a single stage of a trellis diagram maps naturally onto the SE graph and, hence, the minimum area layout techniques previously developed in the literature are of great interest. The shuffle-exchange graph consists of $n = 2^s$ nodes, each of which correspond to one state of the single-stage trellis. Since the number of nodes is a power of two we will refer to this graph as a 2-SE graph. Each node, labeled from 0 to $n-1$, is associated with an equivalent $\nu$-bit binary string $s_{n-1}, \cdots, s_1, s_0$. Two nodes $K$ and $K'$ are linked via a shuffle edge if $K'$ is a left or right cycle shift of $K$ (i.e., if $K = s_{\nu-1}, s_{\nu-2}, \cdots, s_1, s_0$ then $K' = s_{\nu-2}, \cdots, s_0, s_1, s_{\nu-1}$, or $K' = s_0, s_1, \cdots, s_{\nu-1}$). Two nodes $K$ and $K'$ are linked via an exchange edge if $K$ and $K'$ differ only in the first bit (i.e., if $K = s_{\nu-1}, \cdots, s_1, s_0, s_{\nu}$ then $K' = s_{\nu-1}, \cdots, s_1, s_0$ where $s_0$ is the logical complement of $s_0$).

The SE graph can be related to a single stage of a trellis diagram with recirculation edges in the following way. Assume that in a single stage of a trellis diagram with $2^s$ states, that the states are labeled from 0 to $2^s - 1$, respect-
tively. Each initial state, identified by an equivalent \( v \)-bit binary label \( x_{r-1}, x_{r-2}, \ldots, x_1, x_0 \) maps into two states: \( x_{r-2}, \ldots, x_0, x_{r-1} \) and \( x_{r-2}, \ldots, x_0, x_{r-1} \). Observe that the single trajectory from an initial state to the two final states can be interpreted as consisting of a sequence of two connected paths: 1) a perfect shuffle operation consisting of the mapping from state \( x_{r-1}, x_{r-2}, \ldots, x_1, x_0 \) to state \( x_{r-2}, \ldots, x_0, x_{r-1} \) followed by 2) an exchange operation from state \( x_{r-2}, \ldots, x_0, x_{r-1} \) to state \( x_{r-2}, \ldots, x_0, x_{r-1} \). This is an exchange operation in the sense that node \( x_{r-2}, \ldots, x_0, x_{r-1} \) passes state information to node \( x_{r-2}, \ldots, x_0, x_{r-1} \) while node \( x_{r-2}, \ldots, x_0, x_{r-1} \) (originating at \( x_{r-1}, x_{r-2}, \ldots, x_0 \)) passes information to node \( x_{r-2}, \ldots, x_0, x_{r-1} \). Consequently, the shuffle and exchange sequence of data transfers would be performed for all nodes in the equivalent 2-SE graph every symbol interval.

A cycle of shuffle edges is known as a necklace. Cleaving the architecture into necklaces is the layout technique used to embed the VA into silicon. As an example, consider the single-stage trellis with recirculation edges illustrated in Fig. 4(a). If we view this from a graph theoretic point of view we can take the nodes and edges and rearrange them to create the planar graph in Fig. 4(b). The nodes which serve only as wiring distribution points can then be split and stretched such that Fig. 4(c) is produced. If the bidirectional edges between nodes 0 and 1, 2, and 3 are illustrated as a single solid edge, then the 2-SE layout in Fig. 4(d) is produced. Appendix A has more details on this procedure.

The SE structure is appropriate only for embedding the VA for binary alphabets [18], one example of which is illustrated in Fig. 4(d). The necklace is drawn 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. Each node is an add-compare-select processor with a small amount of control logic, while edges define processor interconnections.

The hardware realization of the necklace in Fig. 4(d) is illustrated in Fig. 5. Note that \( y \), the input signal to the receiver, is fed to a branch metric generator which could be either a small ROM or correlator [22], depending on the application (e.g., adaptive, soft, or hard decision decoding). The corresponding \( \lambda \)'s are delivered to the appropriate adders (perhaps 8-bit adders). Two ACS units are detailed in this layout slice along with registers for the path survivors \( \hat{x} \). The survivor sequence register, implemented as a shift register, is typically capable of holding 5\( r \) symbols, 10 bits in this example. Overflow control of the finite-length state metric register can be achieved by subtracting a constant from all the state metrics [9] or choosing one state metric and subtracting it from all other state metrics each (or perhaps once every few) baud intervals. The two output wires or buses \( (1, \hat{x}) \) from each of the two ACS units are drawn unnecessarily long before connecting to the horizontal buses to emphasize the necklace structure.

To complete the design as illustrated in Fig. 5, two additional ACS units (corresponding to the two self-loops) would have to be added. One ACS unit would be connected to the upper horizontal bus while the other ACS unit would be connected to the lower horizontal bus. The benefits of this layout strategy become increasingly apparent for larger algorithm memory lengths.

Increasing the memory of the algorithm produces the VLSI layout illustrated in Fig. 6(a). The SE layout can be generalized to accommodate \( M \)-ary alphabets [18] in what would be referred to as an \( M \)-SE layout, as illustrated in the 3-SE layout of Fig. 6(b), for the ternary case. In Fig. 6(b), nodes 5, 15 and 19 form a necklace because they are
cyclic shifts of one another when in ternary form as 012, 120, 201, respectively. Nodes 15, 16, and 17 are members of the same exchange edge because for each, the first digit is any valid number within the number radix 3, while all other digits are identical when written in ternary form as 120, 121, and 122, respectively. Note also the three self-loops 0, 13, 26 in this figure. In general, there are $m$ self-loops, $m$ nodes in each exchange edge, and no more than $v$ nodes in each necklace.

Knowing the algorithm memory length $v$ and the alphabet size $m$, the general procedure for producing area-efficient layouts is: 1) Each of the $m^v$ nodes are labeled with a $v$-digit $M$-ary label; 2) associate each of the nodes with the appropriate necklace, by determining those that are cyclic shifts of one another; 3) on a perpendicular unit grid, order the necklaces from left to right such that the minimum value of the nodes in each necklace form an increasing sequence; 4) determine those nodes that belong to same exchange edge by collecting those for which the first digit is any valid number within the radix $m$ while the last $v-1$ digits are identical; 5) move nodes within a necklace so that appropriate exchange edges can be inserted (this may require that some of the necklaces be stretched in the vertical direction); 6) additional movement of the necklaces (while maintaining the same topology) and compaction (along unoccupied grid lines) of the layout can then be done.

Two important properties of the SE approach to laying out the VA, irrespective of the alphabet size, are: 1) that all the necklaces in the layout have precisely $v$ nodes when the algorithm memory length is prime and 2) for any algorithm memory length the maximum vertex degree of the graph is four. Structuring the trellis as a series of necklaces produces an elegant and partitionable VLSI layout. Each necklace could be a VLSI chip or each necklace may be an arrangement of integrated circuits on a set of printed
circuit boards. Other important future developments include wafer-scale integration and the use of optical connections in such a layout strategy [24]–[26].

The second architectural strategy based on the cube-connected cycles, discussed below (see also Appendix B), contains even more interprocessor wire area than the SE layout, but has some interesting applications of its own.

When using the CCC network to support an implementation of the VA, we capitalize on the fact that the CCC network consists of several sets or rows of (pipelined) processors where the outputs from the last set of processors are fed back (recirculated) to the first set of processors. Each row or set of processors (one processor for each state) is used to simulate the action of one of the stages in the trellis diagram. Hence, only one row of processors are active during a symbol interval. The state metrics needed by the row as it simulates the \((k + 1)st\) stage of the trellis are obtained from the previously computed outputs of the \(k\)th stage of computation. The attractiveness of the CCC network is that the wiring scheme is simpler and more regular than the SE network (for large \(v\)) while at the same time processors can deliver information, at the next baud interval, directly to the processor required in the next stage. If the connectivity was not precisely right, during a single baud interval, the state metrics would have to circulate more than one stage through the network of processors to move data to the appropriate processor, thus degrading performance.

As initially proposed, the cube-connected cycles [27] layout is derived from a Boolean hypercube of \(2^v\) vertices by replacing each vertex with a cycle of \(v\) vertices. The CCC structure, with \(v2^v\) nodes directly implements \(v\) stages of the trellis diagram with recirculation, as illustrated in Fig. 7 for the case \(v = 2\). Such a structure can be shown to efficiently embed the VA for binary alphabets; hence, we shall refer to this structure as a 2-CCC graph. Each uniquely labelled node in the trellis is a simple processor. By taking the structure in Fig. 7(a) and physically rearranging the location of processors 5 and 6, the structure in Fig. 7(b) is generated. From this graph, the horizontal edges destined to become cycle connections in Fig. 7(c) can be readily identified. All other transitions in Fig. 7(b) can be generated by an additional routing operation along a cube connection. Cycle connections are illustrated by the vertical wires, cube connections are illustrated by the horizontal wires, in Fig. 7(c). This structure has been shown to have some optimal (minimum) area properties [27] and consequently is of great interest.

With the notation we have chosen, a slightly simpler bus structure and data flow results if we reverse the direction of the data flow through the trellis. Now, data transfers involve transfers around the cycle and across the cube. Using this strategy, Fig. 8 illustrates the hardware realization of the cycle containing nodes 0 and 4 in Fig. 7(c). The
Fig. 8. Data paths within the cycle of Fig. 7. Same nomenclature as Fig. 5. The two nonoverlapping clock signals $\phi_1$ and $\phi_2$ are active alternately, one during even bit intervals, the other during odd bit intervals.

In digital communications, this structure has important applications. As an example, consider the trellis produced by puncturing certain convolutional codes [28], illustrated in Fig. 9. The appropriate computational structure after puncturing the trellis in Fig. 9(a) is immediately recognized in Fig. 9(b) as a CCC structure. Similarly, encoded MSK Viterbi receivers [29], considered in the companion paper, have a two-stage trellis structure corresponding to even and odd-bit intervals. Again, the trellis naturally maps onto a CCC structure. The CCC layout structure has other im-
important applications which are detailed in [18]. Notable among these are: the decoding of multiplexed data streams, the decoding of interleaved convolutional; codes for burst noise channels, and generalizations to M-ary alphabets in the so-called M-CCC structures.

VI. AREA–TIME COMPLEXITY MEASURES

In a VLSI implementation of the VA we are interested in two measures of complexity: the chip area \( A \), and the computation time \( T \) (the reciprocal of the maximum baud rate, i.e., the minimum symbol interval that can be supported). In order to establish lower bounds on the performance of any VLSI implementation of the VA, the model of VLSI computation due to Thompson is adopted.

Performing the VA on a VLSI circuit, which consists of a network of simple interconnected processing elements operating in parallel, involves moving a certain minimum amount of information from one side of the chip to the other, at each baud interval. Intuitively, one would expect that a network with very few wires connecting processors together, though requiring relatively small die area (because of this lack of wiring), would be relatively slow in completing all the required interprocessor data transfers and contingent computations, during a single baud interval. Conversely, a network with perhaps the same number of processors, though with enough wire to move data rapidly across any given partition of the chip, could conceivably handle higher baud rates at the expense of increased die area devoted to interprocessor wiring. In other words, for a fixed logic gate delay, there is an apparent area–time tradeoff available to the designer [30]; fast architectures which support high baud rates (small \( T \)) are necessarily large in die area (large \( A \)), while slow architectures (large \( T \)) can be quite small in area (small \( A \)). Indeed, one can show that for the VA certain functions of an area–time product must be greater than a prespecified parameter, in an asymptotic sense.

By standard bisection flow arguments [17] it can be shown [18] that if a VLSI design with area \( A \) executes the VA with alphabet size \( m \) and memory \( v \) in symbol interval \( T \), then the (area)\*(baud rate)\(^{-2} = AT^2 \) product is \( AT^2 = \Omega(m^{3v/2}) \). From this lower bound it directly follows that for any VLSI implementation of the VA, the energy consumption during each symbol interval (i.e., power) as defined by the \( AT \) measure of complexity is given by \( AT = \Omega(m^{3v/2}) \). The \( AT \) measure can be interpreted as the reciprocal of throughput per unit area. Hence, an architecture which is optimized to achieve this lower bound can claim to have made best use of the chip area.

The existence of these complexity measures indicates that a Viterbi receiver operating at a particular baud rate requires at least a certain minimum amount of chip area (in given technology), because a certain minimum amount of information must be transported, by the wires, from one side of the chip to the other regardless of how clever we are in placement of the digital logic elements on the chip. Establishing the \( AT \) lower bound for the VA is important from two perspectives. 1) It establishes, within the VLSI grid model, necessary (but not sufficient) conditions on the relationship between certain parameters of a VA VLSI circuit. (For example, between baud rate, chip area, algorithm memory length, power dissipation, etc.) Hence, it allows one to verify whether claims made on the resource consumption of a VLSI architecture that implements the VA are in fact true by checking to see whether the lower bound is violated. 2) It allows one to establish how close-known designs approach those best possible even though these best realizations, relative to the criterion at hand, may still be unknown. This provides researchers with a navigational aid for guiding the direction of research activity, in much the same way that coding theorists rely on Shannon's channel-capacity theorem to guide their search for low \( p(e) \) codes.

VII. CONCLUSION

A taxonomy of VLSI processor architectures are identified in this paper for channel and source decoding operations in a Viterbi algorithm-based receiver. Classified in
terms of increasing interprocessor wire area they are: uniprocessor, shuffle-exchange, and cube-connected cycle layouts.

The results presented are important from several perspectives. 1) They complement the persistent efforts in literature directed at embedding other types of dynamic programming algorithms in silicon (e.g., recognition of context-free languages, dynamic time warping of speech signals, etc.). 2) Good practical VLSI layouts are generated, by a design methodology, in which we have theoretical assurances that the layout is optimal with respect to an area criterion. 3) The concurrent VLSI architectures presented directly apply to decoding (or detecting, as applicable):

- M-ary signals on intersymbol interference (ISI) channels,
- binary convolutional codes (CC),
- dual-K codes (M-ary CC),
- punctured CC,
- interleaved CC for burst-noise channels,
- CC combined with MLSE on ISI channels,
- partial-response signals,
- coded modulation with multilevel/phase signals,
- correlative encoded MSK (minimum shift keying), and
- multi-h phase codes.

In addition, asymptotic lower bounds on the product

\[
(area) \cdot (\text{baud rate})^{-1}\quad \text{and}\quad (area) \cdot (\text{baud rate})^{-2}
\]

were presented, for which any VLSI implementation of the VA must obey.

### APPENDIX A

**THE SHUFFLE-EXCHANGE NETWORK**

The purpose of this Appendix is to informally provide additional background material on the shuffle-exchange (SE) network organization. We pursue the development of the 2-SE organization throughout, to illustrate concepts. In the following paragraphs we move towards getting a mathematical description of such a network and ultimately to a VLSI grid model representation.

A processing network consists of a collection or group of interconnected processors. There are two popular interconnection strategies for boosting the efficiency of a processing network: 1) pipelining and 2) 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. The shuffle-exchange interconnection network is an example of a recirculation network. When using the SE network to support an implementation of the VA, each stage of the computation uses this row of processors (one processor for each state) to simulate 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 previously computed outputs of the \(k\)th stage of computation. The attractiveness of using the SE network is that processors can deliver information, at the next baud interval, to the processor required without passing that information through intermediate processors. If the connectivity was not precisely right, during a single baud interval, the state metrics would have to circulate more than once through the network of processors to move data to the appropriate processor, thus degrading performance. (The finite time required for this routing operation is one of the major factors that imposes an upper bound to the maximum baud rate that could be supported by such a computational network.)

An eight-node shuffle-exchange network is shown in Fig. 10(a). The concept of a processor is equated to that of a node (in the graph theoretic sense). We assume that the number of nodes, \(n\), is a power of two, and hence, refer to this SE network as a 2-SE network to remind us of this assumption. Suppose these \(n\) processors are numbered 0, 1, \(\cdots\), \(n - 1\), respectively. The exchange operation consists of even-numbered processors swapping or exchanging information with their next higher odd-numbered neighbor. Thus, using a wire or equivalently, an edge, 0 is connected to 1, 1 to 0, 2 to 3, 3 to 2, and so on. The shuffle operation consists of each processor \(i\) transferring information, using an edge, to processor \(2i \pmod{n - 1}\). The shuffle interconnection derived its name from the following analogy. Consider a deck of \(n\) cards, numbered 0, 1,
perpendicular grid lines. The smallest bounding rectangle processors is large. Though the edges may correspond to (log \_n), unit squares (where required interconnect can be performed along unoccupied, its utility is found in the insight it gives us in the placement model, this bounding rectangle must contain at least cn^2/area, to within a constant factor. It has been shown that, containing the total network is an estimate of the final die area of only one vertex point, for an object which presumably at first glance since the processors are allocated an area of only one vertex point, for an object which presumably will require a significant amount of die area. However, its utility is found in the insight it gives us in the placement and routing required to create compact layouts. The need for such insight is acutely apparent when the number of processors is large. Though the edges may correspond to buses and the nodes to finite area processors, the grid model provides a structured design methodology which generates good practical layouts, as illustrated in Fig. 10(d).

**APPENDIX B**

**THE CUBE-CONNECTED CYCLES NETWORK**

This Appendix provides additional background material on the cube-connected cycles (CCC) organization. The CCC interconnection network is an example of a processor interconnection strategy which utilizes both pipelining and recirculation techniques to boost efficiency. In the following paragraphs we move towards getting a mathematical description of such a network and ultimately to a VLSI grid model representation.

The cube-connected cycles derived its name from the following analogy. Consider the CCC structure containing 24 processors, illustrated in Fig. 11. Clearly, as drawn, the structure resembles a cube with each of its "corners" replaced by a cycle of three processors. In general, it is a hypercube of dimension v with each of its vertices replaced by a cycle of v processors.

An eight-node cube-connected cycles network is shown in Fig. 12(a). The concept of a processor is equated to that of a node (in the graph theoretic sense). We assume, for binary alphabets in general, that there are v sets (labeled from 0 to v - 1) of 2^v nodes each (labeled from 0 to 2^v - 1), for a total of v 2^v processing modules, and hence, refer to this CCC network as a 2-CCC network to remind us of this assumption. Each of these processors are uniquely identified by the notation p\_ri, where r specifies the set (one of v) and i specifies the location within the set (one of 2^v). The cycle operation consists of processor p\_ri, communicating, over what is known as a cycle connection, with processor p\_{r+1(mod v)i}. A sequence of cycle operations will form a loop containing v processors. The cube connection operation consists of processor p\_ri, communicating, over
**The Cube-Connected Cycles Graph**

$n = \nu 2^r$ nodes

(a) Cube-connected cycle interconnections for $n = 8$

(b) Each node label is replaced by its binary equivalent

(c) The cube-connected cycles layout in the VLSI grid model

(d) A practical implementation where the nodes are processors that occupy finite $O(1)$ area

Fig. 12. The eight-node 2-CCC processor organization.

---

Table I defines the state metrics that each processor evaluates when labelled as in Fig. 12. Note that each node in the network is responsible for evaluating a particular state metric, however, nodes in the same cycle do not necessarily evaluate the same state metric. Since only one row of processors can be active during each symbol interval, corresponding to the one active stage of trellis, only one processor in a cycle will ever be active. Consequently, Table I indicates that one half of the nodes are active during even baud intervals while the other half are active during odd baud intervals.

There are certain conventions that have developed in the literature with regard to illustrating such a network. The $P_i$ notation is often abbreviated by referring to a single binary label formed by the concatenation of the two respective binary strings “r” and “i” or by the subsequent decimal equivalent of this single binary string. The two layers of evenly spaced wires, horizontal and vertical correspond, respectively, to cube and cycle connections. Cube connections, though bidirectional, are usually drawn as a single solid edge. Edges are often undirected, since we are primarily concerned only with connectivity. Constructing such a graph can be accomplished quickly by generating each of the $2^n$ cycles first and then connecting the cycles together by adding the appropriate cube edges at each node, as illustrated in Fig. 12(c).

Embedding the CCC network within the VLSI grid model involves placing nodes at the vertices of a rectangular, unit grid such that the required interconnect can be performed along perpendicular grid lines. It has been shown that the best possible layout must occupy at least $\nu n^2/(\log_2 n)^2$ unit squares (where $c$ is a positive constant), no matter how clever we are at arranging the nodes and edges, if it contains $n$-node 2-CCC network drawn using the VLSI grid model. For a given number of nodes $n$, the area required by the layout is identical (in an asymptotic sense) to that of the SE, though for a given value of $\nu$ the SE requires less area since it has fewer nodes.

As for the SE in Appendix A, the final CCC construct in Fig. 12(c) may appear rather artificial at first glance since the processors are allocated an area of only one vertex point, for an object which presumably will require a significant amount of die area. However, the design methodology generates good practical layouts, as illustrated in Fig. 11(d), even though the edges may correspond to buses and the nodes to finite-area processors.
ACKNOWLEDGMENT

P. G. Gulak would like to thank Stanford University for providing the atmosphere and facilities to further develop these ideas. The authors would also like to thank the reviewers for providing many useful suggestions that were helpful in improving the clarity of this paper.

REFERENCES