Chapter 33

Decoding of Rate k/n Convolutional Codes in VLSI [1]

V. P. Roychowdhury [2]
P. G. Gulak [2]
A. Montalvo [2]
T. Kailath [2]

Abstract

A systematic procedure for an efficient VLSI implementation of the Viterbi algorithm for decoding convolutional codes is presented. This implementation is based on a network of simple processors, each performing an add-compare-select and branch-generation operation, that reside on a single die and are connected to execute the Viterbi algorithm in a highly parallel way. The chip area of such implementations will depend on the processor interconnections, which in turn depend on the state transition diagram of the convolutional encoder (or dual encoder). It is shown that for all rate $1/n$ convolutional codes generated by feed-forward FIR encoders the encoder state transition diagram, which is described by a de Bruijn graph, can be mapped by a simple equivalence relation to a well-known interconnection scheme in parallel processing referred to as the shuffle-exchange network, for which layout techniques that achieve a proven lower bound on implementation area in a VLSI medium have been established. These results are then extended

[1] This work was supported in part by the National Science Foundation under Grant DCI-84-21315-A1, the U. S. Army Research Office under Contract DAAL03-86-K-0045, the SDIO/IST, managed by the Army Research Office under contract DAAL03-87-k-0033 and Rockwell International Contract INT 6G3052.

[2] Stanford University, Stanford, CA.

to rate $1/n$ codes generated by (IIR) encoders containing feedback. Finally, in the case of general (feed-forward and feedback) rate k/n convolutional encoders it is shown that the state transition diagram of either the encoder or the dual encoder can be always mapped to the Cartesian product of de Bruijn graphs, and therefore of shuffle-exchange graphs; the point is that optimum VLSI layouts for the Cartesian product are easier to obtain and much less complicated than any direct VLSI layouts for the original state transition diagram.

33.1 Introduction

Optimal (maximum likelihood) decoding of convolutional codes can be accomplished by a Viterbi algorithm based on the state transition diagram of the encoder. The Viterbi algorithm (VA) [Vit67], [VO79], [For73b] is in essence a technique for estimating the state sequence of a finite state Markov process observed in memoryless noise. Central to the VA is the concept of a trellis diagram, which is a graphical representation of the state diagram drawn as a function of discrete time. There are q^v nodes at each time step in the trellis diagram, where q is the alphabet size and v is the total number of memory elements in the encoder (often referred to as the constraint length). The VA can be thought of as a dynamic programming solution to the problem of finding the shortest path in this trellis diagram [Omu69]. The essence of the algorithm is a relatively simple procedure of add, compare, and select operations that must be applied to each of the q^v nodes belonging to the same time step in the trellis diagram. The sole reason that the VA is computationally demanding is that the number of operations during each symbol interval T (defined as the time interval between two consecutive output symbols of the receiving channel) grows exponentially with the constraint length of the encoder. It has been traditionally implemented on a single processing element driven by a control unit (*e.g.* a microprocessor) using a direct sequential algorithm. This approach requires $O(q^{v+1})$ operations and $O(q^v)$ random accesses to the processor's memory during each symbol interval T. The throughput rate of such an implementation may not be acceptable in applications where high data rates are required.

The only alternative to increasing the throughput rate is to perform the operations in parallel, thus increasing the hardware complexity. Fortunately, the advent of VLSI (Very Large Scale Integration) technology has opened up opportunities for realizing the parallelism inherent in computationally intensive algorithms. The increase in hardware complexity may now be tolerable as the VLSI technology is capable of realizing chips with the hundreds of thousands of transistors required to realize the VA for constraint lengths of commercial interest. This observation has led to several attempts at providing parallel implementations of the VA in VLSI (see *e.g.*, [CK84], [GS86], [CY86], [Gul84]). Gulak and Shwedyk [GS86], [Gul84] described a fully parallel implementation of the VA based on a set of q^v processors connected according to a *shuffle-exchange* network, for which area efficient VLSI

layouts are known in the literature [KLLM81], [Lei81] (there is a restriction in [GS86] that the encoder be a rate $1/n$ feed-forward (FIR) circuit). It should be pointed out 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.*, [PV81], [Ull81] and thus the VA (for rate $1/n$ FIR encoders) can be efficiently implemented on any of these architectures. We should remark that among these architectures, the shuffle-exchange networks have the least VLSI area for the same number of nodes.

Another family of parallel implementations was presented by Chang and Yao [CY86]. 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 $O(q^v/v)$. In fact the gain in speed over that of the sequential processor can be shown to be at best qv. Hence, though an exponential number of processors is used, the gain in throughput rate is at best logarithmic. This is in sharp contrast to the fully parallel (shuffle-exchange) implementations of Gulak and Shwedyk [GS86], [Gul84] where the gain in speed is directly proportional to the number of processors used. Of course, a price is paid for speed in terms of the area required to layout the architectures in VLSI. In fact, if N is the total number of processors used, then the area required by the fully parallel version is $O(N^2/\log^2 N)$ whereas the area required by the systolic implementation is $O(N)$. In [GK], we have studied other non-fully-parallel architectures that may often be preferred to linear systolic architecture (especially a so called 'cascade' architecture); however, in the rest of this paper we shall confine ourselves to fully parallel implementations. One reason is that despite the results mentioned above, several questions remain unanswered for fully parallel implementations. First, efficient VLSI implementations are not known for the k/n convolutional codes with or without feedback. Even for the rate $1/n$ case, VLSI implementations are known only for the feed-forward encoders. Moreover, the issue of optimality of the VLSI implementations (*e.g.*, those appear in [GS86], [Gul84]) have not been addressed.

In this paper we provide answers to several of these open questions. First we prove the important fact (see Section 33.2.1) that for efficient and fully parallel implementation of the VA, the processors must always be connected according to the state transition diagram of the encoder. Thus, for efficient implementation in VLSI, it will be desirable to devise layouts for encoder state transition diagrams for which both the area and the average length of interconnecting wires is minimum (for faster communication and less cost). We show that the state diagrams of rate $1/n$ feed-forward encoders, which are known as de Bruijn graphs, can be efficiently laid out in VLSI by modifying existing layouts for shuffle-exchange networks. We then show that the state transition diagram of a rate $1/n$ encoder *with feedback* is either

a de Bruijn graph or a subgraph of such a graph, when the encoder realization is in a certain (controller) canonical form. Thus, for *any* rate $1/n$ encoder the VA can be efficiently implemented in VLSI on a set of processors connected according to a de Bruijn or equivalently a shuffle-exchange network. For rate k/n feed-forward encoders with certain 'obvious' realization (to be defined in section 33.3), we show that the state diagrams can be represented as Cartesian product of k, possibly distinct, de Bruijn graphs. Minimum area VLSI layouts for the product graphs are presented using a recursive layout technique that uses the optimal layout strategy for shuffle-exchange networks. For the general case of rate k/n encoders (*i.e.* with and without feedback) it is shown that the dual encoder always has the feed-forward structure for which the state diagram can be efficiently represented as a product graph. In such cases it is recommended that one should apply the VA on the trellis diagram corresponding to the dual encoder, since the state diagram of the original encoder may be arbitrarily complex and efficient VLSI implementation may not be possible. Thus in general, the VA for rate k/n encoders can be always implemented in parallel on a set of processors connected according to a Cartesian product of de Bruijn (or shuffle-exchange) graphs.

The rest of the paper is organized as follows. In section 33.2 we briefly discuss the Viterbi algorithm and introduce the relevant terminologies. We also present an equivalence between de Bruijn graphs and shuffle-exchange graphs that enables us to use the existing layouts for shuffle-exchange networks for efficiently laying out de Bruijn networks. Techniques to generalize such results so as to provide efficient implementation when the encoder has feedback are also outlined. Section 33.3 deals with rate k/n encoders. First it is shown that the state diagram for feed-forward rate k/n encoders can be represented as a Cartesian product of smaller de Bruijn graphs. Optimal layouts for such product graphs are presented and a comparison with the direct implementation is made. It is shown that the product graph representation saves an exponential factor in silicon area for only a constant factor loss in the throughput rate. Space limitations prevent us from providing rigorous proofs for several theoretically intriguing results, nonetheless they are presented in this section for completeness (the reader is referred to [RGMK87] for more details). Finally, section 4 contains some concluding remarks.

33.2 VLSI Architectures for Decoding Rate 1/n Convolutional Codes

A rate $1/n$ convolutional encoder is a finite-state linear sequential circuit operating on GF(q) that has one input and n outputs (see [For70], [For73a]). The number of memory elements (or shift registers) in the circuit is called the constraint length of the encoder. For example Fig. 33.1 shows a rate 1/2 convolutional encoder with constraint length 2.

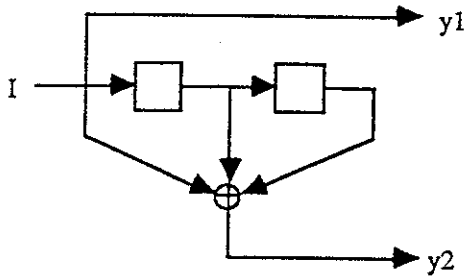


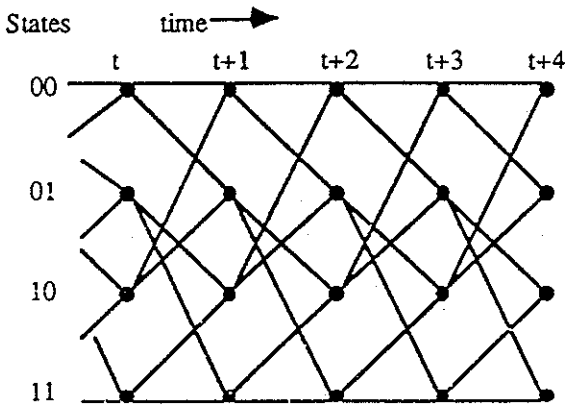
Figure 33.1: A rate 1/2 convolution encoder.

33.2.1 The Viterbi Algorithm

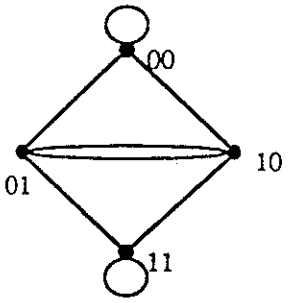
The VA [Vit67] was originally invented for decoding convolutionally encoded data, though since then, it has been applied to other types of problems (*e.g.*, the trellis-coded modulation schemes used in high-speed data modems). The basic theory behind the VA is readily available in the literature and for a good survey the reader is referred to Forney [For73b]. In this section we shall introduce the basic concepts that are required for describing its implementation in VLSI.

The VA can be thought of (see [Omu69]) as a dynamic programming solution to the problem of estimating the state sequence of a finite-state Markov process observed in memoryless noise. Central to the VA is the concept of the *trellis diagram*, which is a graphical representation of the state diagram of the encoder drawn as a function of discrete time. Each time step corresponds to a single symbol (or baud) interval T, and corresponds to one stage of the trellis. The number of stages in the trellis diagram is equal to the length of the input data sequence. The number of nodes (or states) at each stage of the trellis is q^v where q is the cardinality of the input alphabet set and v is the constraint length of the encoder. Each node at every stage of the trellis diagram represents one possible state of the encoder. There is an edge between the node S_i^t (*i.e.*, the node representing state S_i at stage t) to 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. Each such edge (*i.e.*, $S_i^t \rightarrow S_j^{t+1}$) is assigned a weight, λ_{ij}^{t+1} (called the *branch-metric*), and is a measure of the 'unlikelihood' that the channel output at time $t+1$ is caused by the state transition $S_i \rightarrow S_j$ in the encoder. Fig. 33.2 gives the state transition and trellis diagrams for the rate 1/2 encoder of Fig. 33.1.

The VA can now be described as follows. Two quantities, namely, the path metric and the survivor sequence are associated with each state of the trellis diagram. The path metric P_j^t of the state S_j at time t is the weighted length of the shortest weighted path (the weights on the edges in the trellis diagram being the branch-



(a) Trellis diagram



(b) State transition diagram

Figure 33.2: The trellis diagram and the state diagram of the convolution encoder in Figure 33.1.

metrics) between the starting node S_0^0 to the node S_j^t in the trellis diagram. Similarly, the survivor sequence Q_j^t for state S_j at time t is the state sequence associated with the shortest weighted path in the trellis diagram between the initial node S_0^0 and the node S_j^t. Once every baud interval, the path metrics are updated as follows:

$$P_j^{t+1} = \min(P_i^t + \lambda_{ij}^{t+1}) \quad \forall\, i \text{ such that } S_i \rightarrow S_j \tag{33.1}$$

where $S_i \rightarrow S_j$ implies that there is a valid state transition from state S_i to S_j and $P_0^0 = 0$. 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 sufficiently long time L, (see *e.g.*, [VO79] where the issue of how large L should be for sufficiently low probability of error is addressed) the survivor sequence of the state with the minimum path metric is chosen to be the estimate for the state sequence of the encoder; one can then complete the decoding procedure by determining the input sequence corresponding to the estimated state sequence.

It is evident that a fully parallel implementation of the VA can be realized by assigning a single processor for every node in the trellis diagram. However, for a message of length L, such a realization will have Lq^v processors (and Lq^{v+1} interconnections); and, moreover, because only q^v processors are active at any time step, such a realization is very inefficient. A somewhat more practical strategy is to use q^v processors connected according to the state transition diagram of the encoder. The processor M_j, at step t, contains the path metric P_j^t and the survivor sequence Q_j^t for state S_j. At step $t+1$ it receives the channel output y_{t+1} and the path metric P_i^t from each of its predecessor state S_i and also computes λ_{ij} for each state transition $S_i \rightarrow S_j$. It then computes P_j^{t+1}, as given by (33.1), and updates the survivor sequence. Since each state has q predecessors, the total time taken at each step (and thus the symbol interval T) is $O(q)$. Hence, a gain of $O(q^v)$ is achieved over a sequential processor (T is $O(q^{v+1})$ for sequential implementation) by using q^v processors in parallel. The major focus of the rest of the paper is to provide efficient VLSI implementations for this architecture. An important measure of efficiency in VLSI technology is the silicon area required to layout an architecture. Apart from the obvious increase in cost due to a larger area, the increased area compounds cost by drastically reducing the yield and by possibly increasing the energy required for communicating information in the system. Thus, to be able to choose the most efficient implementation, among a host of possible choices, one needs a model that can quantify the notion of 'area' of a VLSI layout. One such popular model, proposed by Thompson [Tho80], is described next.

33.2.2 VLSI Grid model

The VLSI grid model proposed by Thompson [Tho80] is simple and assumes that the chip consists of a grid of vertical and horizontal tracks, spaced apart by some

unit interval. Processors are viewed as points on the grid and are located only at the intersections of grid tracks. Wires are routed through the tracks in order to connect pairs of processors. Wires can intersect only at right angles and overlapping is not allowed. From a computational point of view this model is attractive since the layout area is calculated easily as the product of the numbers of vertical and horizontal tracks in the layout. We should comment that a major motivation behind this model is the observation that the total area in VLSI is often dominated by the wiring rather than the processors; that is why the processors are assumed as just points.

33.2.3 de Bruijn and Shuffle-Exchange Graphs

The state at time t of an encoder can be represented by a v-tuple

$$S^t = < x_{v-1}\ x_{v-2}\ \cdots x_0 >$$

where x_i is the content of the i^{th} memory element. For a feed-forward encoder the next state S^{t+1} will then be given by the tuple

$$S^{t+1} = < x_{v-2}\ x_{v-3}\ \cdots x_0\ a > \quad \forall\, a \in \{0, \cdots, q-1\}$$

where q is the alphabet size. For $q = 2$, the state transition graph generated by the above transitions is known as the de Bruijn graph or Goods diagram of order v (see [Gol82]). For the rest of the section we shall restrict ourselves to the binary alphabet (*i.e.*, $q = 2$) case, primarily because the results are simpler to state and can be easily extended for arbitrary alphabet size q (see *e.g.*, [Gul84], [RGMK87]). In our interpretation, an edge of a de Bruijn graph is called a *shuffle edge* if it is of the form $< x_{v-1}\ x_{v-2}\ \cdots x_0 > \rightarrow < x_{v-2}\ x_{v-3}\ \cdots x_0\ x_{v-1} >$ and it is called a *shuffle-exchange* edge (for reasons to be explained later) if it is of the form $< x_{v-1}\ x_{v-2}\ \cdots x_0 > \rightarrow < x_{v-2}\ x_{v-3}\ \cdots x_0\ \overline{x_{v-1}} >$. An 8 node de Bruijn graph representing the state diagram of encoder with constrain length 3 is shown in Fig. 33.3.

de Bruijn graphs are closely related to shuffle-exchange graphs, which are well-known in parallel processing (see *e.g.*, [Sto71], [Ull81]). A *shuffle-exchange* graph of

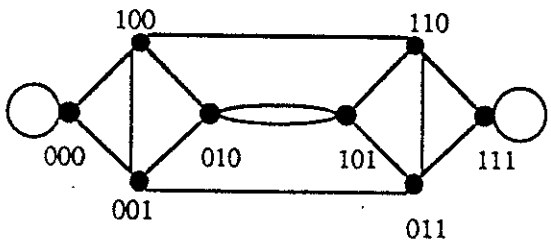


Figure 33.3: An 8 node de Bruijn graph.

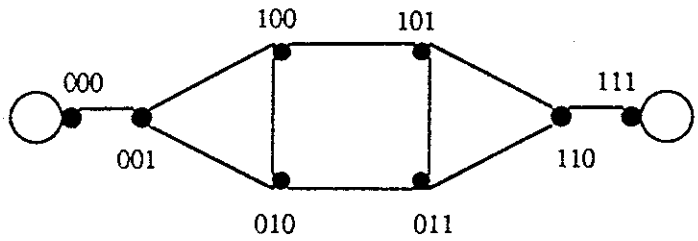


Figure 33.4: An 8 node shuffle-exchange graph.

order v has 2^v nodes, each labeled by a v tuple $< x_{v-1}\ x_{v-2}\ \cdots x_0 >$. Every node $< x_{v-1}\ x_{v-2}\ \cdots x_0 >$ is connected by a *shuffle* edge to the node $< x_{v-2}\ \cdots x_0\ x_{v-1} >$ and by an *exchange* edge to the node $< x_{v-1}\ x_{v-2}\ \cdots \overline{x_0} >$. A cycle of shuffle edges is known as a necklace. An 8 node shuffle-exchange graph is shown in Fig. 33.4.

Several authors (see *e.g.*, [Lei81], [KLLM81]) have presented good layout techniques for shuffle-exchange networks. These techniques cleave the network into necklaces and then appropriately insert the exchange edges. Thompson [Tho80] showed that the minimum VLSI layout area for a N-node shuffle-exchange graph is $\Omega(N^2/\log^2 N)$ and then Kleitman *et al.* [KLLM81] devised optimal layout techniques that achieve the lower bound. We shall show next that these optimal layouts of shuffle-exchange networks can be easily modified to yield optimal layouts for de Bruijn networks.

33.2.4 Optimal Layouts for de Bruijn Networks

The layouts for de Bruijn graphs can be easily motivated by redrawing the graph so that it looks exactly like a shuffle exchange graph of the same order. The shuffle edges can be drawn just as they are in de Bruijn graphs; however, a shuffle-exchange edge is redrawn by first routing it along a shuffle edge (*i.e.*, along $< x_{v-1}\ x_{v-2}\ \cdots x_0 > \rightarrow\!\!< x_{v-2}\ x_{v-3}\ \cdots x_0\ x_{v-1} >$) and then along an exchange edge (*i.e.*, along $< x_{v-2}\ x_{v-3}\ \cdots x_0\ x_{v-1} > \rightarrow\!\!< x_{v-2}\ x_{v-3}\ \cdots x_0\ \overline{x_{v-1}} >$); (Remark: this also justifies the term 'shuffle-exchange' edge). The resulting graph looks exactly like a shuffle-exchange graph, except that it has two parallel edges for every edge in the latter graph (see Fig. 33.5). This simple procedure can be used to modify any layout of a shuffle-exchange graph to a layout of a de Bruijn graph; the resulting layout will have two wires for every wire in the original layout. Hence, the length and the width of the new layout will be at most twice the original values. Thus, the area of the de Bruijn graph layout is at most 4 times the area of the corresponding shuffle-exchange network. A rigorous proof can be found in [RGK87], [RGMK87].

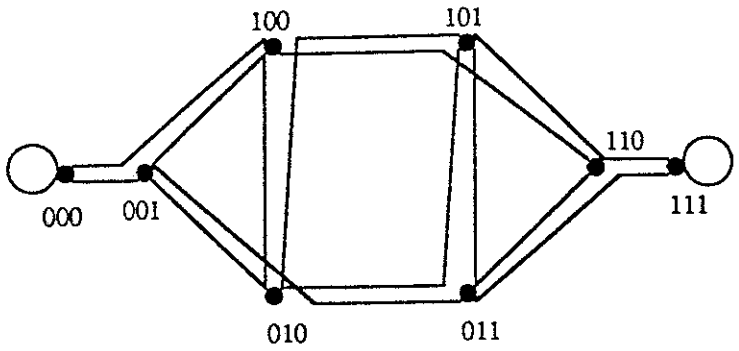


Figure 33.5: An 8 node de Bruijn graph redrawn as a shuffle-exchange graph.

Thus, we can obtain $O(N^2/\log^2 N)$ area layouts for de Bruijn networks by modifying the optimal layouts for shuffle-exchange networks. Since, the minimum area for de Bruijn graphs is $\Omega(N^2/\log^2 N)$ ($\Omega(q^2N^2/\log^2 N)$ for arbitrary alphabet size q, see [RGK87]), the layouts obtained by modifying the optimal layouts for shuffle-exchange graphs are also optimal for de Bruijn graphs. [There is an erroneous claim in Samatham and Pradhan [SP84] that the area for de Bruijn graphs is $\Omega(N^2/\log N)$] .

33.2.5 Architectures for decoding rate $1/n$ codes with feedback

Decoding for encoders with feedback can also be done on a set of processors connected according to a de Bruijn graph of appropriate order; hence, the efficient layout strategies discussed before can be reapplied. First we shall define a canonical structure for encoders with feedback. Let x_i^t denote the content of the i^{th} memory module (or shift-register) at time t. Then, a rate $1/n$ encoder is said to be in the controller canonical form if the memory elements can be ordered such that $x_i^{t+1} = x_{i-1}^t \; \forall \; i = 1, \cdots, \; v-1$ and $x_0^{t+1} = f(u, x_0, x_1, \cdots, v_{v-1})$, where $f(.)$ is some function defined on GF(q) (see Fig. 33.6).

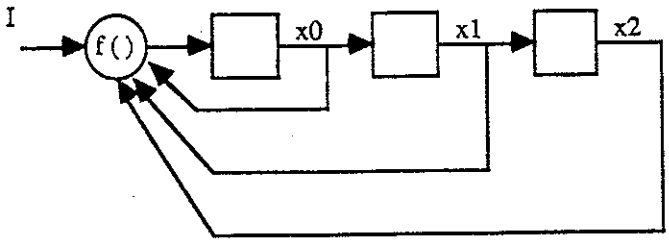


Figure 33.6: A rate 1/n encoder realized in the controller canonical form.

Theorem 1 *The state diagram of a controller canonical form is either a de Bruijn graph or a subgraph of such a graph.*

Proof: We can label a state as a v-tuple $< x_{v-1}\ x_{v-2}\ \cdots x_0 >$. If an edge is in the state diagram, G, of the encoder then it is of the form $< x_{v-1}\ x_{v-2}\ \cdots x_0 > \rightarrow < x_{v-2}\ x_{v-3}\ \cdots x_0\ f(.) >$. However, $f(.)$ is either 0 or 1 (assuming $q = 2$), which means that the edge is also in the de Bruijn graph of order v. Hence, G is contained in the de Bruijn graph of order v. □

Since, convolutional encoders are linear systems in GF(q), it can be shown that (see [Kai80], [For70]) any rate $1/n$ encoder can be realized in the controller canonical form and hence can be implemented on a set of q^v processors connected according to a de Bruijn graph.

33.3 VLSI Architectures for Decoding Rate k/n Convolutional Codes

A rate k/n convolutional encoder is a finite-state linear sequential circuit operating on GF(q) that has k inputs and n outputs. We shall first discuss decoders for feed-forward rate k/n codes and then extend the results to codes with feedback.

A feed-forward encoder can be always realized in an obvious manner as shown in Fig. 33.7 (see [For70]). In the obvious realization, each input has a separate sequence of shift-registers associated with it (*i.e.*, the input I_i has a shift-register bank of length v_i). The outputs are given as linear functions of the internal states and the inputs. The total state of the encoder at time t can then be given by

$$S^t =< s_1^t, s_2^t, \cdots, s_k^t >$$

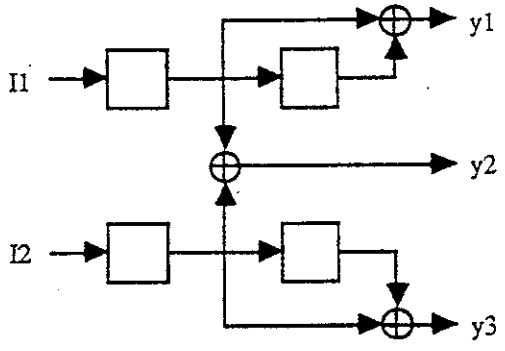


Figure 33.7: A rate 2/3 encoder realized in the obvious manner.

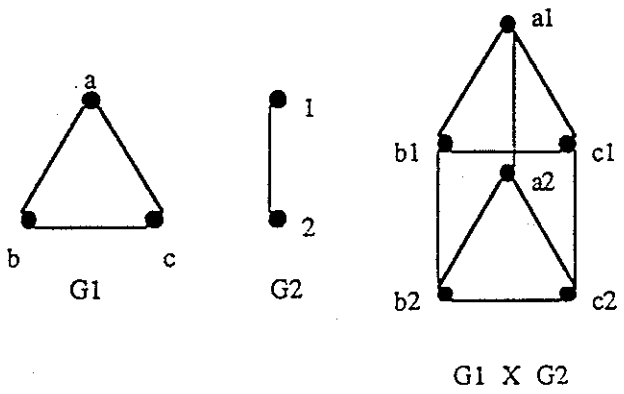


Figure 33.8: An example of Cartesian product of two graphs.

where s_i^t is the state of the shift-register sequence associated with the input I_i and can itself be represented by the tuple $s_i^t =< x_{v_i-1}\ x_{v_i-2} \cdots x_0 >$. The next state of the feed-forward encoder is then given by

$$S^{t+1} =< s_1^{t+1}, s_2^{t+1}, \cdots, s_k^{t+1} >$$

where s_i^{t+1} is the next state corresponding to the state s_i^t and can be represented by the tuple $s_i^{t+1} =< x_{v_i-2}\ x_{v_i-3}\ \cdots x_0\ a >$ $\forall\ a\ \in \{0, \cdots,\ q-1\}$. One can now draw the state transition diagram for the encoder as given by the above transitions (see Fig. 33.8). It has q^{v+k} nodes, (where $v = \sum_1^k v_i$) and each node has degree $2q^k$. We know that the decoder can be always realized efficiently on a set processors connected according to the state transition diagram of the encoder (see Section 33.2.1). However, efficient layout techniques for such state transition diagrams are not readily apparent.

33.3.1 Product Graph Representation of The State Transition Diagram

However, there is an alternative representation of the state transition diagram that is easy to layout and requires considerably less area. In the new representation, a state transition $S^t \rightarrow S^{t+1}$ is done in k steps, where in the i^{th} step only the state s_i^t changes to s_i^{t+1} and the rest of the states remain unchanged. The resulting representation of the state transition diagram can be formulated as a *Cartesian product* of the state state transition diagrams of the individual shift-register banks, *i.e.*,

$$G = G_k \times G_{k-1} \times \cdots \times G_1 \qquad (33.2)$$

where G_i is the state transition diagram of the i^{th} shift-register bank (hence, a de Bruijn graph of order v_i) and the Cartesian product of two graphs is defined as

follows (see Figure 33.8 for an example of Cartesian product graphs):

Definition 1 *The Cartesian product of two graphs* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *is the graph* $G = (V, E)$ *where* $V = V_1 \times V_2$ *and an edge* $< (u_1, u_2), (v_1, v_2) > \in E$ *if and only if either* $u_1 = v_1$ *and* $< u_2, v_2 > \in E_2$ *or* $u_2 = v_2$ *and* $< u_1, v_1 > \in E_2$.

The reason that the modified state transition diagram is given by (33.2) follows directly from the above definition; a rigorous proof is given in [RGMK87]. We should note here that in the product graph representation one has to make k transitions in order to obtain a valid state transition of the encoder; the degree of every node in this representation is $2qk$ and the total number of edges is $kq^{\nu+1}$.

Efficient Layouts for Product Graphs

A convenient way of interpreting $G = G_1 \times G_2$ is that G can be obtained by replacing every node of G_1 by a copy of G_2 and then interconnecting these macro-nodes according to the interconnection pattern of G_1. This suggests a recursive VLSI layout procedure: 1) layout G_2 optimally (since every individual graph is de Bruijn, we can apply the results of previous section), 2) layout copies of G_2 and connect them in the same way as the nodes of G_1 are connected in an optimal layout of G_1. The procedure can easily continue for any $k > 2$. This intuitive recursive layout technique turns out to be also asymptotically optimal and achieves the lower bound $\Omega(q^2N^2/v_{max}^2)$ derived in [RGMK87] (v_{max} is the length of the longest shift-register bank, and N is the total number of nodes in the graph). The lower bound on the VLSI area for the state transition diagram is shown to be $\Omega(q^{2k}N^2/v_{max}^2)$ in [RGMK87]. Thus, the product graph representation saves a factor of q^k in the silicon area. Since one has to make k transitions in the product graph to make a valid state transition, it can be shown that the symbol interval increases by at most a constant factor; however, the reduction in area outweighs the loss in speed. Furthermore, it can be shown that the product graph representation requires only q input and q output ports for every processor, whereas the direct implementation requires q^k such ports. If the processors are required to have a fixed number of ports, then one can show that the direct realization of the state transition diagram is no longer even faster.

33.3.2 Architectures for decoding rate k/n codes with feedback

The state transition graph will not be nicely structured for an encoder with arbitrary structure. However, for every encoder one can define a dual encoder (see [For73a], [For70]) and the VA based on the state transition diagram of the dual encoder

can be used to decode channel outputs without any loss of information. It turns out that the dual encoders are always feed-forward and hence the decoder can be implemented on a set of processors connected according to the product graph of de Bruijn graphs of appropriate orders.

33.4 Concluding Remarks

In this short paper we have described some parallel architectures that are suitable for efficient implementations in VLSI and can execute the VA for decoding convolutional codes. They are shown to be related to well-known architectures for parallel processing such as shuffle-exchange and de Bruijn networks. More detailed accounts of the material presented here can be found in [RGMK87] and [RGK87].

References

[CK84] J. B. Cain and R. A. Kriete. A VLSI r=1/2, k=7 Viterbi Decoder. *Proc. of NAECON*, May 1984.

[CY86] C. Y. Chang and K. Yao. Systolic Array Processing of the Viterbi Algorithm. *Submitted to IEEE Transactions on Information Theory*, June 1986.

[For70] G. D. Forney. Convolutional Codes I: Algebraic Structure. *IEEE Transactions On Information Theory*, IT-16, No. 6:720–738, Nov. 1970.

[For73a] G. D. Forney. Structural Analysis of Convolutional Codes via Dual Codes. *IEEE Transactions on Information Theory*, IT-19:512–518, July 1973.

[For73b] G. D. Forney. The Viterbi Algorithm. *Proceedings of The IEEE*, 61, No. 3:268–278, March 1973.

[GK] P. G. Gulak and T. Kailath. Locally Connected VLSI Architectures for the Viterbi Algorithm. *IEEE Journal on Selected Areas in Communications*, to appear in 1988.

[Gol82] S. W. Golomb. *Shift Register Sequences.* Aegan Park Press, 1982.

[GS86] 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.

[Gul84] P. G. Gulak. *VLSI Structures For Digital Communications.* PhD thesis, University of Manitoba, Winnipeg, Canada, Dec. 1984.

[Kai80] T. Kailath. *Linear Systems.* Prentice-Hall Inc., Englewood Cliifs, N.J., 1980.

[KLLM81] 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*, 278–292, May 1981.

[Lei81] 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.

[Omu69] J. K. Omura. On the Viterbi Algorithm. *IEEE Transactions Information Theory*, IT-15:171–179, Jan. 1969.

[PV81] 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.

[RGK87] V. P. Roychowdhury, P. G. Gulak, and T. Kailath. Optimal VLSI layouts of de Bruijn Graphs. *to be submitted to IEEE Trans. on Computers*, 1988.

[RGMK87] V. P. Roychowdhury, P. G. Gulak, A. Montalvo, and T. Kailath. Decoding of Rate k/n Convolutional Codes in VLSI. *to be submitted to IEEE Trans. on Info. Theory*, 1988.

[SP84] M. R. Samatham and D. K. Pradhan. A Multiprocessor Network Suitable for Single-chip Implementation. *Proc. 11th Ann. Symp. on Computer Architecture*, 328–337, June 1984.

[Sto71] Harold S. Stone. Parallel Processing with the Perfect Shuffle. *IEEE Transactions On Computers*, c-20, No. 2:153–161, Feb. 1971.

[Tho80] C. D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Dept. of Comp. Science, Carnegie Mellon University, Pittsburgh,PA, 1980.

[Ull81] J. D. Ullman. *Complexity of VLSI Design.* MIT Press and John Wiley Sons, Inc., 1981.

[Vit67] A. J. Viterbi. Error Bounds for Convolutional Codes and an asymptotically optimum decoding algorithm. *IEEE Transactions Information Theory*, IT-13:260–269, Apr. 1967.

[VO79] A. J. Viterbi and J. K. Omura. *Principles of Digital Communication and coding.* New York: McGraw Hill. 1979.