Area-Efficient Architectures for the Viterbi Algorithm—Part II: Applications

C. Bernard Shung, Horng-Dar Lin, Robert Cypher, Paul H. Siegel, and Hemant K. Thapar

***Abstract*—In the previous paper, we established the theoretical foundations of a new class of area-efficient architectures for the Viterbi algorithm. In this paper, we will show area-efficient architectures for practical codes to illustrate the design procedures and demonstrate the favorable area–time tradeoff results. Three examples from convolutional codes, matched-spectral-null (MSN) trellis codes, and Ungerboeck codes will be presented. We will also discuss the application of our area-efficient techniques to codes with a very large numbers of states, codes with time-varying trellises, and a programmable Viterbi decoder.**

I. Introduction

In the previous (Part I) paper, we established the theoretical foundations of a new class of area-efficient architectures for the Viterbi algorithm. In this paper, we will show area-efficient architectures for three practical examples to illustrate the design procedures and demonstrate the favorable area–time tradeoff results. The first example is a 16-state de Bruijn graph trellis in convolutional codes. The second example is a six-state matched-spectral-null (MSN) trellis code used in partial response channels. The third example is a 16-state Ungerboeck code used in the coded modulation. We will also discuss the application of our area-efficient techniques to codes with a very large numbers of states, codes with time-varying trellises, and a *programmable* Viterbi decoder.

II. Application Examples

A. de Bruijn Graphs

Trellises with the structure of a de Bruijn graph often occur in convolutional codes [5].[1] Fishburn and Finkel [2] found that a de Bruijn graph (they called it a *four-pin shuffle*) of 2^M states can be emulated by a smaller de Bruijn graph of 2^L states, where $M > L$. The partitioning suggested in [2] can be called *MSB grouping,* where states with the same L most significant bits are partitioned into the same ACS. However, issues such as scheduling and local memory implementation were not addressed.

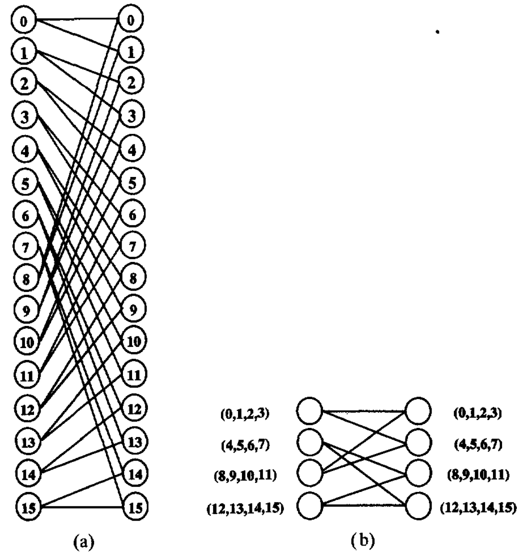

Fig. 1. (a) A 16-state de Bruijn graph trellis. (b) A four-state de Bruijn graph trellis which emulates the trellis in (a) in a totally uniform manner.

We found that the LSB scheduling (by their $M - L$ least significant bits) allows a simple SPN implementation. Let $N = 2^M$, $P = 2^L$; from Theorem 2 in the previous paper, we know that the matrix permutation reduces to within-block permutations and a fixed interconnection. For de Bruijn graphs, the within-block permutation has two special characteristics that can be exploited to simplify the local memory design. First, the permutations are *shuffle* permutations. In a shuffle permutation of size $N/P = 2^{M-L}$, the state i with binary address $i_{M-L-1} \cdots i_1 i_0$ will be permuted to $i_0 i_{M-L-1} \cdots i_1$. Furthermore, the same d metrics are required in d consecutive time units. Hence, only one SPN is needed, and the d metrics can be obtained by tapping off at d adjacent pipeline registers. *Selective latches* (SL's) have to be used to latch in d new data every d cycles and retain the values for $d - 1$ cycles.

Each shuffle permutation of size 2^{M-L} can be implemented by a special-purpose SPN that consists of $M - L - 1$ serial shuffles of size $2^{M-L-2}, \cdots, 2, 1$ (see the Appendix). This efficient implementation has a significant impact on the fea-

Paper approved by the Editor for VLSI in Communications of the IEEE Communications Society.
Manuscript received August 29, 1990; revised April 14, 1991. This paper was presented in part at the IEEE Global Telecommunications Conference, San Diego, CA, December 1990.
C. B. Shung is with the Department of Electronics Engineering, National Chiao-Tung University, Hsinchu, Taiwan 30039.
H.-D. Lin is with AT&T Bell Laboratories, Holmdel, NJ 07733.
R. Cypher is with the Almaden Research Center, IBM Corporation, San Jose, CA 95120.
P. H. Siegel is with the Signal Processing and Coding Division, IBM Corporation, San Jose, CA 95120.
H. K. Thapar is with the Storage Systems Department, Products Division, IBM Corporation, San Jose, CA 95120.
IEEE Log Number 9209480.

[1] Strictly speaking, a de Bruijn graph corresponds to the trellis of a rate $1/N$ feedforward convolutional code. We are considering de Bruijn graphs with *modified* branch labels.

0090-6778/93$03.00 © 1993 IEEE

sibility of our area-efficient architectures on the de Bruijn graph trellises. The latency of this SPN is $\beta \simeq 2^{M-L-1} = (1/2)(N/P)$. If $\alpha = (N/P) - \beta \simeq (1/2)(N/P) \leq \bar{\alpha}$, then we have a constant factor of two slowdown! If $(1/2)(N/P) > \bar{\alpha}$, then the slowdown factor is $1 + (1/\bar{\alpha})(1/2)(N/P)$. In both cases, the factor of slowdown is much smaller than the factor of area saving.

Let us use a 16-state trellis as an example, which is shown in Fig. 1(a). A four-state trellis that emulates the 16-state trellis totally uniformly is shown in Fig. 1(b). The S, X, and Y matrices of such a partitioning and scheduling of $N = 16$ and $P = 4$ are shown below.

$$S = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix}$$

$$X = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \\ 12 & 13 & 14 & 15 \end{pmatrix}$$

$$Y = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 8 & 8 & 9 & 9 \\ 2 & 2 & 3 & 3 \\ 10 & 10 & 11 & 11 \\ 4 & 4 & 5 & 5 \\ 12 & 12 & 13 & 13 \\ 6 & 6 & 7 & 7 \\ 14 & 14 & 15 & 15 \end{pmatrix}.$$

An overall pipelined area-efficient architecture is shown in Fig. 2(a)–(c) for the 16-state trellis.[2] The timing diagram is in Fig. 3. It is a factor of two slower than the full parallel implementation, but the area saving is roughly a factor of three (not four considering all the overhead.) Based on the above analysis, we know that the area–time tradeoff will be increasingly favorable to the area-efficient architecture as N becomes larger.

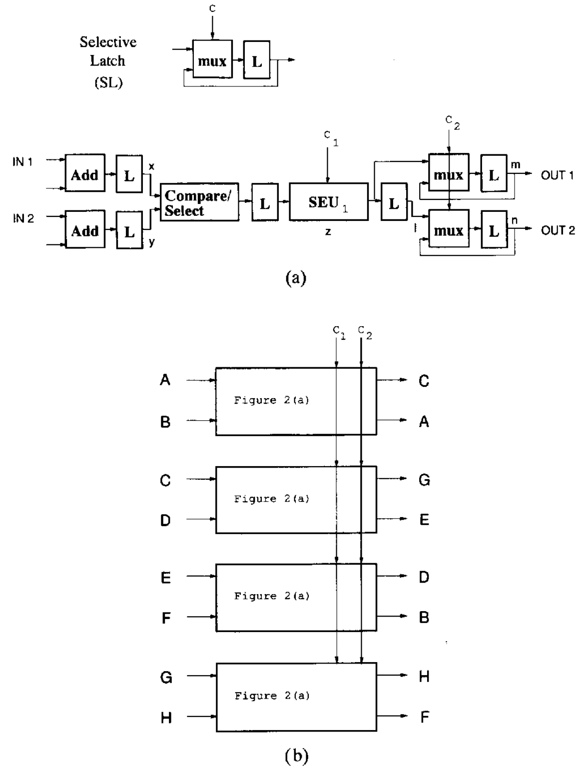

Fig. 2. (a) The area-efficient architecture of one pipelined ACS with local memory for the 16-state de Bruijn graph trellis. To support the with-block permutation, an SEU of size 1 (controlled by C_1) and two selective latches (SL) (controlled by C_2) are used. The total latency is four, which is the same as the number of states sharing the ACS. The letters x, y, z (internal latch in the SEU), l, m, and n denote the content of the pipeline latches, respectively, and will be referred to in the timing diagram in Fig. 3. (b) The complete architecture which contains four copies of (a). The letters A–H indicate the feedback connections. Note that this is a fixed-interconnection network.

B. Matched-Spectral-Null Trellis Code

In this section, we used a nonhomogeneous rate 8/10 MSN trellis code [4] [Fig. 4(a)] as an example, which is useful in partial response channels. Our architecture can be extended to MSN trellis codes with other rates.

A trellis of $P = 2$ can be constructed [Fig. 4(b)] to emulate the original trellis. Note that each connection contains two parallel branches, and hence the degree of ACS1 and ACS2 is 4. The partitioning of states is done by *edge grouping:* each branch in the $P = 2$ trellis emulates a set of branches in the $N = 6$ trellis that have the same branch label. States 0, 2, and 4 are partitioned to ACS1; states 1, 3, and 5 are partitioned to ACS2. We choose to put in three pipeline stages in the ACS's, one after the adder and one after each of the compare-selects. We need at least one extra pipeline stage because we need to bring out two metrics at a time. Therefore, we have one pipeline bubble because the number of pipeline stages is four while the number of states sharing one ACS is three. Consequently, two *dummy* states, 6 and 7, are put in. State 6 is partitioned to ACS1 and state 7 is partitioned to ACS2.

It can be seen that states 1 and 2 require the same set of path metrics (from states 0, 1, 2, and 3) so we schedule them in the same time unit. Likewise, states 3 and 4 are scheduled in the same time unit. States 0 and 7 are scheduled in the same time unit because state 7, being a dummy state, can take an arbitrary set of path metrics which is required by state 0. Likewise, states 5 and 6 are scheduled in the same time unit. The partitioning and scheduling are summarized by the S matrix below.

[2] In this example, the branch metric computation is not discussed because it depends on the particular convolutional code.

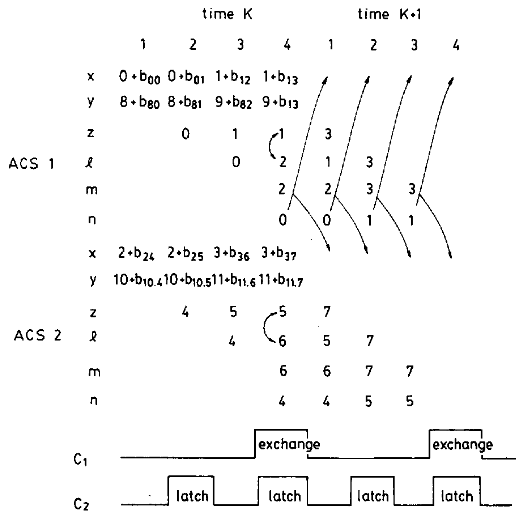

Fig. 3. The timing diagram of the area-efficient implementation in Fig. 2(a), (b). Each major cycle contains four minor cycles, and the contents of the pipeline registers at each minor cycle are shown. When C_1 is HIGH, the contents in z and l are exchanged. When C_2 is HIGH, the contents of z and l are latched into the selective latches (SL's). The arrows indicate how the new path metrics are used in the next minor cycle. Their uniformity verifies the fixed interconnection.

$$S = \begin{pmatrix} 0 & 2 & 4 & 6 \\ 7 & 1 & 3 & 5 \end{pmatrix}$$

$$X = \begin{pmatrix} 0 & 2 & 4 & 6 \\ 0 & 2 & 4 & 6 \\ 0 & 2 & 4 & 6 \\ 0 & 2 & 4 & 6 \\ 7 & 1 & 3 & 5 \\ 7 & 1 & 3 & 5 \\ 7 & 1 & 3 & 5 \\ 7 & 1 & 3 & 5 \end{pmatrix}$$

$$Y = \begin{pmatrix} 6 & 0 & 2 & 4 \\ 0 & 2 & 4 & 6 \\ 7 & 1 & 3 & 5 \\ 1 & 3 & 5 & 7 \\ 6 & 0 & 2 & 4 \\ 0 & 2 & 4 & 6 \\ 7 & 1 & 3 & 5 \\ 1 & 3 & 5 & 7 \end{pmatrix}$$

From the X and Y matrices above, we observe that: 1) the within-column permutations are the same for all four time units, thus allowing a fixed-interconnection realization; and 2) the within-row permutations are only circular shifts which can be implemented by tapping at different points in the pipeline registers. Therefore, neither an MIN nor an SPN is required in the area-efficient architecture.

The detailed circuit block diagram of the area-efficient design is shown in Fig. 5. Note that the tapping points in ACS1 and ACS2 are different. In ACS1, we tap off after the fourth and fifth pipeline registers (net A and C). In ACS2, we tap off after the third and fourth pipeline registers (net B and D). This offset, however, will cause the path metrics of the wrong frame to be accessed at the first and last time unit. Fortunately, because states 0 and 5 only need two out of the four input metrics, they can be scheduled at the first and last time unit, respectively, to avoid the problem. Two additional control signals (C_1 and C_2 in Fig. 5) are used to turn off the effect of the path metrics of the wrong frame.

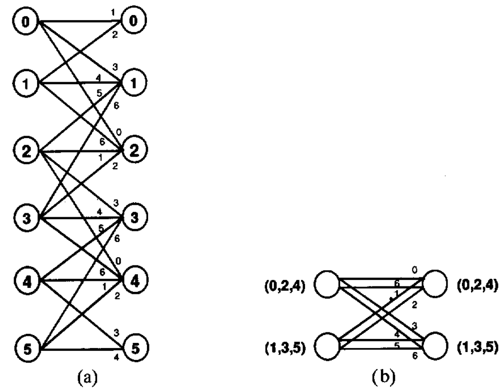

Fig. 4. (a) A six-state, nonhomogeneous trellis from a rate 8/10 matched-spectral-null (MSN) trellis code. (b) A two-state, degree-4, homogeneous trellis which emulates the trellis in (a) (not in a totally uniform manner). *Edge grouping* is apparent by comparing (a) and (b).

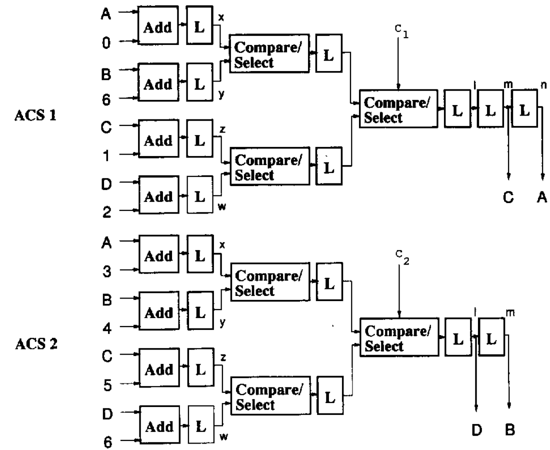

Fig. 5. The area-efficient implementation for the six-state MSN trellis code. The letters $A-D$ indicate the feedback connections. The numerals 0–6 indicate the branch metrics. C_1 and C_2 are used to turn off half of the ACS for states 0 and 5.

If not for the time inefficiency due to the dummy states, the pipelining would have made up the speed loss from sharing ACS's. Therefore, our 2-ACS design is only 33% slower than a state-parallel 6-ACS design. From six ACS's to two pipelined ACS's we save roughly 50% in area because two out of the six ACS's are smaller and the pipeline registers contribute to a larger ACS area than that of a nonpipelined one. In addition, the wiring area is also reduced significantly. This is due to: 1) fewer interconnections between ACS's (bandwidth efficiency), and 2) the branch metrics need to be routed to only one ACS (by virtue of edge grouping). Overall, our area-efficient design provides a very favorable area–speed tradeoff, which has been

verified by a chip implementation of the rate $8/10$ MSN trellis code [7].

C. Ungerboeck Codes

Coded modulation with multilevel/phase signals, first proposed by Ungerboeck [8], has proven to be an effective technique in communications. The structure of Ungerboeck codes can be viewed as de Bruijn graphs with M-ary alphabets, and the branch labels are obtained by the *set partitioning* principle. Both characteristics are found to be very useful in the area-efficient architectures.

In this section, we will use a 16-state Ungerboeck code for coded 8-PSK modulation as an example (see Fig. 6) [8]. Each state is degree-4. The 16 states are partitioned into two ACS's by *edge grouping:* half of the states take 0, 2, 4, 6 as branch labels, and the other half takes 1, 3, 5, 7 as branch labels. The S, X, and Y matrices are shown below.

$$S = \begin{pmatrix} 0 & 1 & 2 & 3 & 8 & 9 & 10 & 11 \\ 4 & 5 & 6 & 7 & 12 & 13 & 14 & 15 \end{pmatrix}$$

$$X = \begin{pmatrix} 0 & 1 & 2 & 3 & 8 & 9 & 10 & 11 \\ 0 & 1 & 2 & 3 & 8 & 9 & 10 & 11 \\ 0 & 1 & 2 & 3 & 8 & 9 & 10 & 11 \\ 0 & 1 & 2 & 3 & 8 & 9 & 10 & 11 \\ 4 & 5 & 6 & 7 & 12 & 13 & 14 & 15 \\ 4 & 5 & 6 & 7 & 12 & 13 & 14 & 15 \\ 4 & 5 & 6 & 7 & 12 & 13 & 14 & 15 \\ 4 & 5 & 6 & 7 & 12 & 13 & 14 & 15 \end{pmatrix}$$

$$Y = \begin{pmatrix} 0 & 8 & 4 & 12 & 10 & 2 & 14 & 6 \\ 4 & 12 & 0 & 8 & 14 & 6 & 10 & 2 \\ 8 & 0 & 12 & 4 & 2 & 10 & 6 & 14 \\ 12 & 4 & 8 & 0 & 6 & 14 & 2 & 10 \\ 1 & 9 & 5 & 13 & 11 & 3 & 15 & 7 \\ 5 & 13 & 1 & 9 & 15 & 7 & 11 & 3 \\ 9 & 1 & 13 & 5 & 3 & 11 & 7 & 15 \\ 13 & 5 & 9 & 1 & 7 & 15 & 3 & 11 \end{pmatrix}$$

ACS1 is to produce updated path metrics for states 0, 1, 2, and 3 in the first four time units. All of them require the same set of path metrics (from states 0, 4, 8, and 12). In matrix Y, their order is deliberately permuted to match the order of the branch metrics. Therefore, two four-input MIN's are required to perform within-column permutations.

The block diagram of the area-efficient design is shown in Fig. 7. The four-input ACS is the same as the one used in the MSN trellis code, which has three pipeline stages. The required SPN's degenerate to just two SEU's of size 2, which are used to reorder $\{0,1,2,3,8,9,10,11\}$ into $\{0,1,8,9,2,3,10,11\}$ and $\{4,5,6,7,12,13,14,15\}$ into $\{4,5,12,13,6,7,14,15\}$. Like the de Bruijn graphs in section A, eight selective latches (SL's) are used to latch the path metrics at the one out of four time units and retain the values at the other three time units. The eight path metrics are divided into two groups, and each is sent to a four-input MIN.

The total number of pipeline stages is 11, with three in the four-input ACS, three in the SEU, three in latches, one in the SL, and one in the four-input MIN. With eight states sharing one ACS, this is not time-efficient. Specifically, the speed is

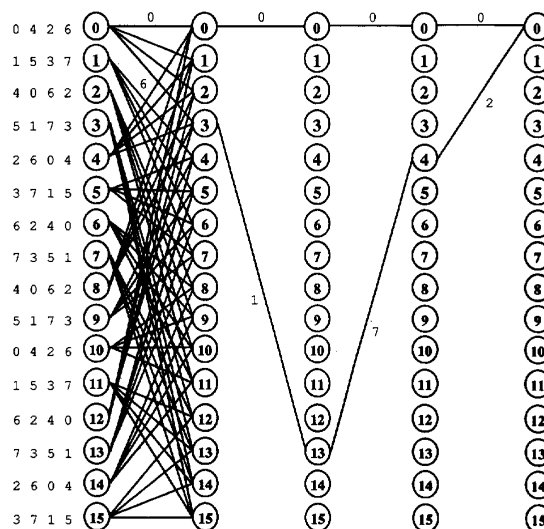

Fig. 6. The trellis diagram of a 16-state Ungerboeck code for 8-PSK [8].

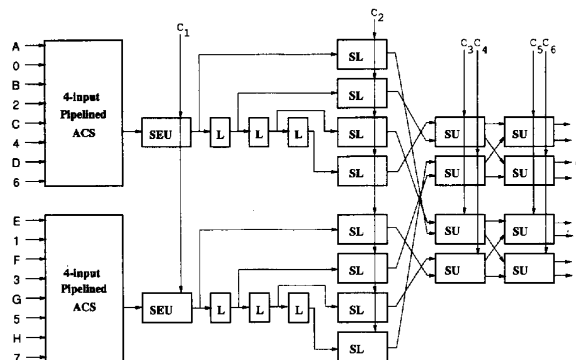

Fig. 7. The area-efficient architecture for the 16-state Ungerboeck code. The selective latch (SL) is illustrated as part of Fig. 2 (a). The switch unit (SU) is illustrated in Fig. 4 in the accompanying (Part I) paper. The letters $A-H$ indicate the feedback connections. The numerals $0-7$ indicate the branch metrics. C_1 controls the two SEU's, C_2 controls the eight SL's, C_3-C_6 control the two four-input MIN's.

$(\alpha+\beta)/\alpha = 11/3$ times slower than that of a state-parallel design. The area saving is estimated to be a factor of eight. A good part of the area saving comes from the interconnection of the branch metrics, which can now be sent to only one ACS, as opposed to being switched in a complicated way in the state-parallel design. Our area-efficient architecture again provides a favorable area–time tradeoff in this example.

In the above table, we summarize the results of the three design examples. Specifically, the ACS row shows the ratios of the number of ACS's used in the state-parallel (SP) and area-efficient (AE) designs. The Area row shows the ratio of the area estimates of SP and AE designs, which takes into account: 1) the ACS number, 2) the routing area, and 3) the area increase by the introduction of pipeline registers in ACS's. The Time row shows the ratio of the cycle time estimates of SP

	de Bruijn (SP):(AE)	MSN (SP):(AE)	Ungerboeck (SP):(AE)
ACS	16 : 4	6 : 2	16 : 2
Area	3 : 1	2 : 1	8 : 1
Time	1 : 2	1 : 1.33	1 : 3.67

and AE designs. It can be seen that in all three examples, the area reduction more than compensates for the speed penalty.

III. Other Applications

Area-efficient architectures are very suitable for codes with a very large number of states. One such project has been undertaken at the Jet Propulsion Laboratory for decoding the data from the *Galileo* spacecraft [1], which involves a low-rate convolutional code with 2^{14} states. A state-parallel implementation has been planned which employs a modular architecture such that the entire Viterbi decoder can be constructed by a few types of different components. From section A, we feel that our area-efficient architecture can be applied to greatly reduce the hardware complexity without much degradation in speed.

Continuous speech recognition with a hidden Markov model (HMM) has been a hot research area recently. It involves implementing the Viterbi algorithm with a large number of states in the trellis (on the order of thousands) at a relatively lower speed (on the order of thousands of samples per second) [6]. Systolic architectures [3] have also been proposed for both the training and recognition phases of HMM. However, they are less effective when the transition in the trellis is sparse (i.e., each state makes a transition to a small portion of all states), which is usually the case in continuous speech recognition.

Our area-efficient architecture matches the continuous speech recognition problem well by providing a means to trade speed for area saving. The HMM at the *word* level is more regular such that some heuristics may be used to achieve an efficient architectural mapping. The HMM at the *grammar* level, however, is less regular, and hence a random matrix permutation may have to be used.

Another application of the area-efficient architectures is the design of a programmable Viterbi decoder. It provides a universal hardware platform, and allows the implementation of arbitrary codes to be done by software. A programmable Viterbi decoder is possible because of the extensive programmability inherent in the SPN and MIN. Once the general-purpose area-efficient architecture is implemented for a particular $\{N, P, d\}$, changing the trellis structure is simply done by changing the matrix permutation. Any trellis with smaller N can also be realized. In this case, the *long-jump* connection in the SPN provides an easy way for the smaller trellis to avoid the large latency built in for the large trellis. It is not possible, however, to map trellises with larger N or d to an architecture designed for a smaller trellis. *Time-varying* codes can also be implemented by the programmable Viterbi decoder. In this case, an additional area saving is obtained because we do not have to design separate special-purpose hardware.

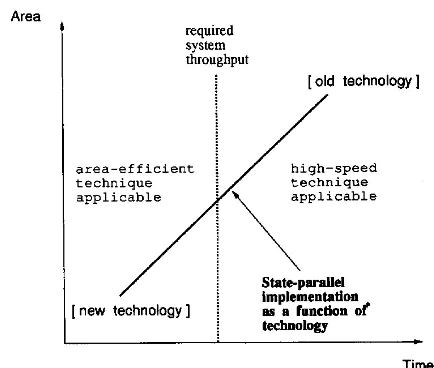

Fig. 8 With the improvement of technology, area-efficient techniques will become more and more useful.

IV. Conclusion

In this paper, we applied the area-efficient techniques developed in the previous (Part I) paper to three practical examples —convolutional codes, matched-spectral-null (MSN) trellis codes, and Ungerboeck codes—and showed favorable results. We also discussed the application of our area-efficient techniques to codes with very large numbers of states, codes with time-varying trellises, and a *programmable* Viterbi decoder.

The authors feel that the area-efficient techniques will become more useful as the technology improves. This argument is explained by in Fig. 8, in which a line is used to represent the state-parallel implementation as a function of technology. For a given required system throughput which corresponds to a vertical line in Fig. 8, it can be seen that the high-speed techniques are required for speed improvement in old technologies, while the area-efficient techniques are required for area reduction in new technologies. One such example is the use of a very-high-speed technology, such as GaAs, where area is at a premium and the circuit speed is expected to be much higher than the desired throughput rate.

Appendix
Shuffle Permutation by SPN

In this Appendix, we show a recursive algorithm for shuffle permutation that is suitable for efficient SPN implementation. In a shuffle permutation of size 2^M, the state i with binary address $i_{M-1} \cdots i_1 i_0$ will be permuted to $i_0 i_{M-1} \cdots i_1$. The following recursive algorithm can easily be proved by induction.

Algorithm: To shuffle 2^M points, do

1) If $M = 1$, stop.

2) For $j = 2^{M-2}+1$ to 2^{M-1}, exchange point j with point $j+2^{M-2}$. There will be 2^{M-2} exchanges altogether. This can be done by an SEU of size 2^{M-2}.

3) Break the 2^M points at the center to form two 2^{M-1} points. Recursively call this algorithm to shuffle each group. □

We will use a 16-point example (see Fig. 9) to illustrate the operation. In the first step, four exchanges occur (4 with 8, 5 with 9, 6 with 10, and 7 with 11). Then the 16 points are

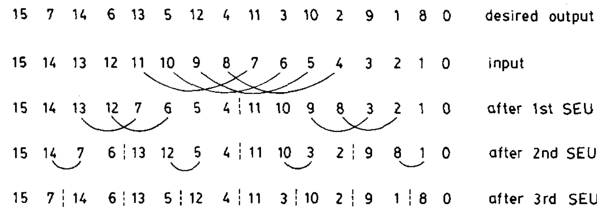

Fig. 9. A 16-point shuffle permutation example in the Appendix.

divided (by the dotted line) into two eight-points groups, each of which undertakes two exchanges in the second step. Note that the two groups can share the same SEU of size 2 because they are shifted through the SEU in sequence. Therefore, the 16-point shuffle permutation can be implemented by an SPN with three SEU's with sizes 4, 2, and 1. In general, to shuffle 2^M points, a special-purpose SPN with $M-1$ SEU's of sizes $M/4, \cdots, 4, 2, 1$ can be used whose total latency is roughly $M/2$.

REFERENCES

[1] O. Collins, S. Dolinar, R. McEliece, and R. Pollara, "A VLSI decomposition of the DeBruijn graph," Tech. Rep. TDA Progress Rep. 42–100, Jet Propulsion Lab., Feb. 1990.

[2] J. P. Fishburn and R. A. Finkel, "Quotient networks," *IEEE Trans. Comput.,* vol. C-31, pp. 288–295, Apr. 1982.

[3] J. Hwang, J. Vlontzos, and S. Kung, "A systolic neural network architecture for hidden Markov models, " *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 37, pp. 1967–1979, Dec. 1989.

[4] R. Karabed and P. H. Siegel, "Matched spectral null trellis codes for partial response channels, Parts I and II," in *Proc. 1988 Int. Symp. Inform. Theory,* Kobe, Japan, June 1988.

[5] S. Lin and D. Costello, Jr., *Error Control Coding: Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1983.

[6] J. Rabaey, T. Stoelzle, D. Chen, S. Narayanaswamy, R. Brodersen, H. Murveit, and A. Santos, "A large vocabulary real time continuous speech recognition system," in R. Brodersen and H. Moscovitz, Ed., *VLSI Signal Processing, III.* New York: IEEE Press, 1988.

[7] C. B. Shung, P. H. Siegel, H. K. Thapar, and R. Karabed, "Implementation issues for the design of a rate 8/10 trellis code chip for partial response channels," in *Proc. 3rd Workshop ECC,* San Jose, CA, Sept. 1989, pp. 213–225.

[8] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory,* vol. IT-28, pp. 55–67, Jan. 1982.

C. Bernard Shung, for a photograph and biography, see the April 1993 issue of this Transactions, p. 643.

Horng-Dar Lin, for a photograph and biography, see the April 1993 issue of this Transactions, p. 643.

Robert Cypher, for a photograph and biography, see the April 1993 issue of this Transactions, p. 644.

Paul H. Siegel, for a photograph and biography, see the April 1993 issue of this Transactions, p. 644.

Hemant K. Thapar, for a photograph and biography, see the April 1993 issue of this Transactions, p. 644.