Efficient Network Folding Techniques for Routing Permutations in VLSI

Hussein M. Alnuweiri, *Member, IEEE,* and Sadiq M. Sait

***Abstract*—Network folding is a technique for realizing permutations on N elements using interconnection networks with M input (and output) terminals, where $M < N$. A major motivation for network folding is the severely limited number of I/O pins in microelectronic packages, such as VLSI chips or multichip module (MCM) packages. Cost overhead and performance degradation due to off-chip communication as well as long on-chip wires may render implementing otherwise good designs infeasible or inefficient. In this paper, an efficient and systematic methodology is proposed for designing *folded* permutation networks that can route the class of *bit-permute-complement* (BPC) permutations. In particular, it is shown that any folded BPC permutation network can be constructed using only two stages of uniform-size *transpose* networks. This results in highly modular structures for BPC networks. The methodology trades off speed (time), with I/O and chip-area.**

***Index Terms*—VLSI interconnection, permutation routing, network folding, BPC permutations, I/O reduction, area-time tradeoffs, I/O-time tradeoffs.**

I. Introduction

HIGHLY PARALLEL fine-grain computations, can be naturally divided into alternating stages of data processing (computation) and communication. Processing is carried out by arithmetic/logic units or processing elements, and communication is realized by one or more stages of permutation networks. This strategy has been particularly useful in deriving very fast parallel architectures for multidimensional signal and image processing, arithmetic circuits, and sorting [3], [6], [8], [9], [13], [16], [20], [21]. However, the implementation of applications with a large number of inputs can be restricted by several physical attributes of the implementation medium. Such attributes include the limited number of I/O terminals per package (a VLSI chip, a multichip module, or a printed circuit board). Another limitation is the allowable maximum wire length. The magnitude of this limitation will continue to increase as the device feature size continues to decrease. The main consequence of such limitations is that the particular design of parallel networks (particularly, interconnection networks) is shaped by specific packaging parameters (e.g., number of I/O pins, available layout area) and other physical specifications (e.g., wire length). In many cases, different parameters mandate significant changes in the original design or replacing the original network entirely by another design that better fits the specifications. To avoid the problem of redesigning parallel architectures whenever certain packaging parameters change, one must seek a methodologies that allow the design process to be parameterized so that a parallel network of a given size can be laid out in packages of various sized by simple remapping.

Towards achieving the above goals, this paper proposes efficient techniques for folding a large class of permutation networks into networks with smaller number of I/O terminals and smaller area (and consequently, smaller maximum wire-length). We are mainly concerned with the class of *bit-permute-complement* (BPC) permutations [15]. A large number of permutations frequently used in parallel systems and arithmetic circuits fall in this class. Examples include, bit-reversal, shuffle-exchange, K-shuffle, butterfly, vector-reversal, and bit-swap permutations. The paper focuses on a methodology for routing this class of permutations in VLSI, under various I/O, area, and time trade-offs. The resulting VLSI designs can route BPC (bit-permute-complement) permutation of size N, using a chip with N/Q I/O pins, $O(N^2/Q^2)$ area, and $O(wQ)$ time, where w is the word length of the permuted elements and $1 \leq Q \leq \sqrt{N/w}$.

The proposed methodology generalizes the *index-mapping* techniques proposed in earlier papers by one of the authors [2]–[4], and results in compact and modular *folded* VLSI networks. The proposed method exploits the relationship between the input and output indices of a permutation to derive a family of networks which can route the permutation under different I/O-time (and area-time) tradeoffs. This is achieved by decomposing the given permutation into a sequence of permutations which can be implemented more efficiently by the chip I/O schedule. Specifically, the proposed methodology derives a five-stage network for any BPC permutation. The five stages contain two stages of folded networks (i.e., they contain memory cells), and three stages of simple two-terminal nets. The folded networks consists of a number of block transpose networks, while the remaining three stages are reduced-size permutation stages derived from the original BPC permutation by a suitable index-map. The index-mapping scheme overcomes the limited-I/O problem by allowing the elements to be input in a multiple number of phases. The permuted elements are also output over a number of phases. The slow-down caused by the multiple number of phases is compensated for by reducing the amount of off-chip communication (I/O), and by

Manuscript received December 21, 1993; revised December 15, 1994. This work was supported by an NSERC operating grant, by the Center for Integrated Computer Systems Research (CICSR) at the University of British Columbia, and by King Fahd University of Petroleum and Minerals.

H. M. Alnuweiri is with the Department of Electrical Engineering, University of British Columbia, Vancouver, B.C. V6T 1Z4, Canada.

S. M. Sait is with the Computer Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia.

IEEE Log Number 9410837.

an improved utilization of the chip area. We demonstrate our techniques by presenting folded VLSI networks for several useful permutations.

A. VLSI Model

This paper assumes a synchronous VLSI model (propagation delay is independent of wire length), with a semelective-unilocal input/output schedule (each input is received only once, at a prespecified input port) [19], [20]. Also, a word-local model is assumed (the w bits representing one element are input or output through the same set of I/O ports) with I/O pads placed on the border of the chip.

II. The Class of BPC Permutations

We will develop a general methodology for folding BPC permutation networks, which generalizes a preliminary scheme reported in [2]. Since permutations are only concerned with the indices (addresses) of elements, elements will be represented by their indices, and all subsequent permutations and operations will be defined on arrays of indices. In general a (single-stage) permutation network can be described by the permutation

$$\begin{pmatrix} 0 & 1 & \cdots & j & \cdots & N-1 \\ \pi(0) & \pi(1) & \cdots & \pi(j) & \cdots & \pi(N-1) \end{pmatrix}$$

where the pair $j, \pi(j)$ in each column is interpreted as a connection between position (or index) j in the input array and position (or index) $\Pi(j)$ in the output array. More regular permutations can be specifies in terms of the operations performed to transform an index j into an another index $\pi(j)$. The class of BPC permutations [15], can be defined in terms of bit-permuting and bit-complementing operations on the binary representation of the indices. Let $x_{n-1}x_{n-2}\cdots x_0$ be a binary string representing the index (position) of an element in an arrangement of $N = 2^n$ elements.

Basically, a BPC permutation on $N = 2^n$ elements (each represented by an n-bit index) is a bijection Π: $\{0,1\}^n \mapsto \{0,1\}^n$, defined as follows:

$$\Pi(x_{n-1}x_{n-2}\cdots x_0) = \pi(x_{n-1})\pi(x_{n-2})\cdots\pi(x_0).$$

where either $\pi(x_i) = x_j$ or $\pi(x_i) = \overline{x}_j$ for some j, and $\overline{x}_j$ is the complement of bit x_j. Although expressed in a different form, the above definition of BPC permutations is equivalent to the one given in [15].

Any BPC permutation Π can be expressed as the composition of two permutations P and C, where permutation P involves only bit-permute (BP) operations and permutation C involves only bit-complement (BC) operations. We chose to first perform the BC permutation (C) to complement all index bits that need to be complemented, then perform the BP permutation (P) to place all index bits in their final positions (according to the original permutation Π). Using standard functional composition notation, this can be expressed by $\Pi = P \circ C$. The permutation Π can be also realized by $C \circ P$, however, this requires a slight modification of the definition of C. The permutations C and P are defined as follows:

$$C(x_{n-1}x_{n-2}\cdots x_0) = c(x_{n-1})c(x_{n-1})\cdots c(x_0),$$
$$P(x_{n-1}x_{n-2}\cdots x_0) = p(x_{n-1})p(x_{n-1})\cdots p(x_0),$$

where $c(x_j) = \overline{x}_j$ if there exists a j such that $\pi(x_i) = \overline{x}_j$. Also, $p(x_i) = x_j$ if $\pi(x_i) = x_j$ or $\pi(x_i) = \overline{x}_j$. For example, if Π is defined as follows:

$$\Pi(x_5x_4x_3x_2x_1x_0) = x_0\overline{x}_1\overline{x}_2x_3\overline{x}_4\overline{x}_5 \tag{1}$$

then permutations C and P are

$$C(x_5x_4x_3x_2x_1x_0) = \overline{x}_5\overline{x}_4x_3\overline{x}_2\overline{x}_1x_0, \tag{2}$$
$$P(x_5x_4x_3x_2x_1x_0) = x_0x_1x_2x_3x_4x_5. \tag{3}$$

In the following, the decomposition of a BPC permutation into BP and BC permutations will be used to facilitate mapping a given BPC permutation onto a VLSI network with limited I/O pins. The procedure is based on developing separate mappings for the BP and BC permutations, then using a combined VLSI network to realize both permutations.

III. Network Folding

This section presents an index mapping schemes that results in simple folded networks for BPC permutation. The main goal is to map a BPC permutation network of size $N = 2^n$ (i.e., with N I/O terminals) into an equivalent network with N/Q I/O terminals. This mapping will be called *folding*, and Q will be called the (I/O) *reduction-factor*. A folded BPC network has N/Q I/O terminals. Thus, bits from at most N/Q elements can be input or output at a time through the folded network. Thus, the index-mapping procedure views the N-element input or output vectors as a matrix of Q columns with N/Q elements per column. Specifically, Let $M = N/Q$, then an $N-$element vector $V_{(N)} \in F^N$ can be arranged into an $M \times Q$ matrix $V_{M\times Q} \in F^{M\times Q}$ as follows:[1]

$$V_{(N)} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{N-1} \end{bmatrix} \Rightarrow$$

$$V_{M\times Q} = \begin{bmatrix} v_{N-M} & \cdots & v_M & v_0 \\ v_{N-M+1} & \cdots & v_{M+1} & v_1 \\ \vdots & \ddots & \vdots & \vdots \\ v_{N-1} & \cdots & v_{2M-1} & v_{M-1} \end{bmatrix}.$$

In other words, column $i(0 \le i \le Q-1)$ of the above matrix contains elements with indices $i(N/Q), i(N/Q) + 1, \cdots, (i+1)(N/Q) - 1$. The mapping (permutation) of elements from the input vector to the output vector can now be expressed as a mapping of elements from the input matrix to the output matrix. For BPC permutations, each index in the output matrix is obtained by performing the specified bit-permute-complement operations on the bit representation of

[1]The notation F denotes a suitable set of numbers, e.g., real or imaginary. F^N denotes the space of column vectors of size N, and $F^{M\times Q}$ denotes the space of $M \times Q$ matrices.

the corresponding index (i.e., the index at the same position) in the input matrix. As an example, the input and output matrices for the bit-reversal permutation expressed by the mapping (3) above, are shown below for $Q = 4$ (and $N = 64$).

$$\underbrace{\begin{bmatrix} 48 & 32 & 16 & 0 \\ 49 & 33 & 17 & 1 \\ 50 & 34 & 18 & 2 \\ 51 & 35 & 19 & 3 \\ 52 & 36 & 20 & 4 \\ 53 & 37 & 21 & 5 \\ 54 & 38 & 22 & 6 \\ 55 & 39 & 23 & 7 \\ 56 & 40 & 24 & 8 \\ 57 & 41 & 25 & 9 \\ 58 & 42 & 26 & 10 \\ 59 & 43 & 27 & 11 \\ 60 & 44 & 28 & 12 \\ 61 & 45 & 29 & 13 \\ 62 & 46 & 30 & 14 \\ 63 & 47 & 31 & 15 \end{bmatrix}}_{\text{Input index matrix}} \xrightarrow{\text{Bit-Reverse}} \underbrace{\begin{bmatrix} 3 & 1 & 2 & 0 \\ 35 & 33 & 34 & 32 \\ 19 & 17 & 18 & 16 \\ 51 & 49 & 50 & 48 \\ 11 & 9 & 10 & 8 \\ 43 & 41 & 42 & 40 \\ 27 & 25 & 26 & 24 \\ 59 & 57 & 58 & 56 \\ 7 & 5 & 6 & 4 \\ 39 & 37 & 38 & 36 \\ 23 & 21 & 22 & 20 \\ 55 & 53 & 54 & 52 \\ 15 & 13 & 14 & 12 \\ 47 & 45 & 46 & 44 \\ 31 & 29 & 30 & 28 \\ 63 & 61 & 62 & 60 \end{bmatrix}}_{\text{Permuted index matrix}}$$

The permuted index-matrix is obtained by applying the bit-reversal permutation (P) to each index in the input index-matrix. In the input matrix, all elements are in their correct original position. After applying permutation P each element is moved into a new position according to the permutation. For example, the element with index 4 (or binary index 000100_2) is mapped to location 8 in the permuted matrix (since $P(000100_2) = 001000_2$). It should be emphasized that the numbers shown in the above matrices denote the index of each element and not their data value. The intention is to illustrate the relationship between element positions in the input and output matrices. The design methodology for two classes of networks is outlined in the next section. The first class implements the bit-permute permutations (hence called BP-permutation networks), and the second class implements bit-complement permutations (hence called BC-permutation networks). The Final BPC permutation network will merge the two networks.

IV. Folded BC Permutation Networks

This section develops a simple technique for constructing BC-permutation networks under the input schedule described in the previous section. Let $x_{n-1}, x_{n-2}, \cdots, x_0$ be an n-bit number representing the index of an element. The notation R_i^j will be used to denote a string of j bits starting at position i and ending at position $i + j - 1$ of some index string [2], [4]. Using this notation, the string $R_{n-q}^q R_0^{n-q}$ denotes an n-bit index, where R_{n-q}^q denotes the column number of the index (as it appears in the input $(N/Q) \times Q$ matrix), and R_0^{n-q} denotes the row number of the index.

Bit-complement (BC) permutations involve complementing certain index bits without changing their position. Thus, if the permutation C involves bits in R_0^{n-q}, then this will involve exchanging rows of the input matrix. However, if C involves also bits in R_{n-q}^q, then columns of the matrix must be exchanged. Note that exchanging (or, in general, permuting) rows can be easily realized by a permutation network of size N/Q, as illustrated in Fig. 1. However, since input columns are available only one at a time, exchanging columns can not be performed by direct interconnection. The method outlined below handles this problem by decomposing a BC permutation in a sequence of permutations which will first transpose segments of input columns then exchange them.

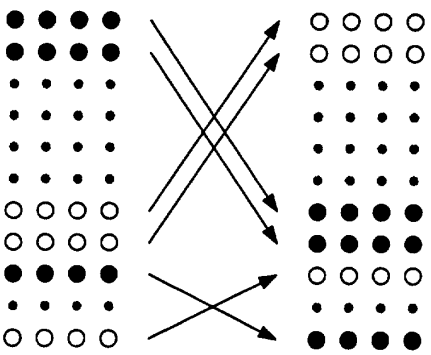


Fig. 1. Exchanging input rows.

Formally, a BC permutation α can be expressed as follows:

$$a, bc \xmapsto{\alpha} \hat{a}, b\hat{c} \tag{4}$$

where a, bc denotes an $n-$bit index such that $a = R_{n-q}^q, b = R_{n-q}^{n-2q}$, and $c = R_0^q$. In this case, the q-bit string a denotes the column index and the concatenated string bc denotes the row index of an element. The strings $\hat{a}, \hat{b}$, and $\hat{c}$, are obtained by complementing certain bits of a, b, and c, respectively, as specified by the permutation α. The above notation places a comma after the bits (or strings) representing the column index. Permutation α can be decomposed into a sequence of four permutations $\alpha = \alpha_4 \circ \alpha_3 \circ \alpha_2 \circ \alpha_1 (\alpha_1$ is applied first) defined as follows.

$$a, bc \xmapsto{\alpha_1} a, \hat{b}\hat{c} \tag{5}$$

$$a, \hat{b}\hat{c} \xmapsto{\alpha_2} \hat{c}, \hat{b}a \tag{6}$$

$$\hat{c}, \hat{b}a \xmapsto{\alpha_3} \hat{c}, \hat{b}\hat{a} \tag{7}$$

$$\hat{c}, \hat{b}\hat{a} \xmapsto{\alpha_4} \hat{a}, \hat{b}\hat{c}. \tag{8}$$

Careful inspection of the above mappings shows that permutations α_1 and α_3 involve row indices only, while permutations α_2 and α_4 swap a column index (a or $\hat{a}$) with a row index ($\hat{c}$). Permutation α_1 complements certain bits of the row index bc as specified by the original permutation α. Thus, α_1 can be realized by direct interconnection (i.e., two-terminal nets). Permutation α_2 swaps the indices $\hat{c}$ and a. Fig. 2 shows the index map for permutation α_2 (see [2], [4] for details on index maps), and an example of applying α_2 to a 16×4 matrix of elements, i.e., $N = 64$ and $Q = 4$, is shown in Fig. 3. For this particular case, each index consists of $n = \log N = 6$ bits[2], and $\alpha_2(x_5x_4x_3x_2x_1x_0) = x_1x_0x_3x_2x_5x_4$.

The index-map indicates that α_2 essentially partitions each column into segments of length Q, then transposes each $Q \times Q$

[2]All logarithms in this paper are base 2.

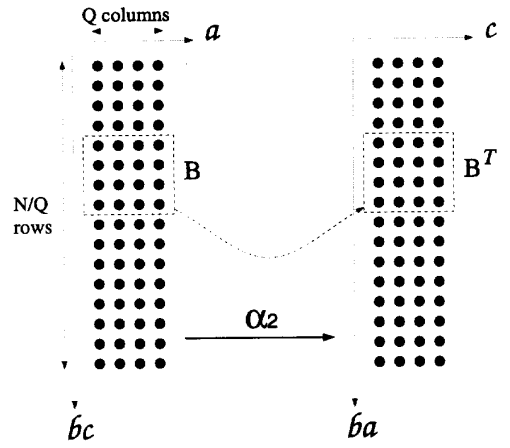


Fig. 2. Index-map of α_2 (B is a $Q \times Q$ array).

48	32	16	0		3	2	1	0
49	33	17	1		19	18	17	16
50	34	18	2		35	34	33	32
51	35	19	3		51	50	49	48
52	36	20	4		7	6	5	4
53	37	21	5		23	22	21	20
54	38	22	6		39	38	37	36
55	39	23	7		55	54	53	52
				$\overset{\alpha_2}{\longmapsto}$				
56	40	24	8		11	10	9	8
57	41	25	9		27	26	25	24
58	42	26	10		43	42	41	40
59	43	27	11		59	58	57	56
60	44	28	12		15	14	13	12
61	45	29	13		31	30	29	28
62	46	30	14		47	46	45	44
63	47	31	15		63	62	61	60

Fig. 3. Index-map of α_2 for $N = 64$, $Q = 4$.

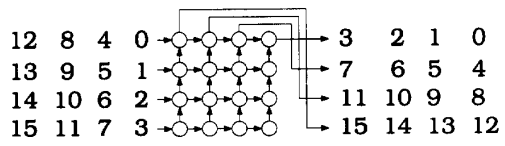


Fig. 4. A 4 × 4 block transpose network.

block of elements (formed by Q segments from Q consecutive columns). Transposing a $Q \times Q$ block can be achieved using a simple transposition network as shown in Fig. 4. This is a *word-model* network which is presumably capable of reading or storing the entire w bits representing an input element simultaneously. Thus each node in Fig. 4 can store w bits and all data paths have width w.

Large word-model networks are not suitable for implementation in VLSI mainly because of their large I/O and area requirements. Fig. 6 shows bit-serial and 2 bit at-a-time

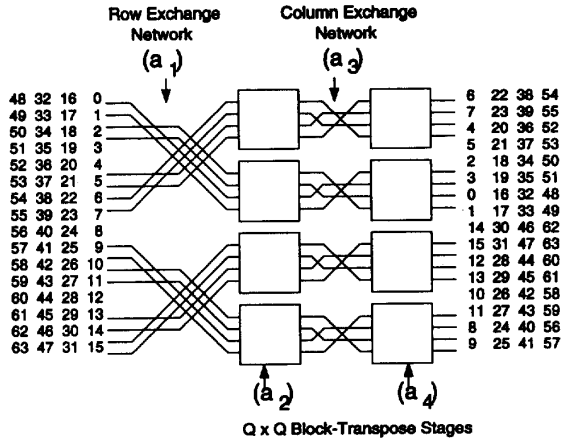


Fig. 5. A BC-permutation network for permutation C.

(2-BAAT) versions of the network of Fig. 4. performance improvements attainable by the bit-serial and r-BAAT approaches are discussed in Section IV. For now, it is sufficient to note that realizing permutation α_2 requires using N/Q^2 block-transpose networks in parallel. The first stage of transpose networks in Fig. 5 constitute a folding network that realizes α_2 for $N = 64$ and $Q = 4$ (i.e., $n = 6$ and $q = 2$).

Permutation α_3 complements certain bits of a as specified by the original permutation α. Since the bits of a are now available as row indices, realizing α_3 can be achieved by direct interconnection (i.e., two-terminal nets). Finally, Permutation α_4 swaps indices $\hat{a}$ and $\hat{c}$, and thus, can be realized by a network identical to the one realizing α_2. Fig. 5 illustrates the basic structure of a complete BC-permutation network by presenting a network for routing permutation

$$C(x_5x_4x_3x_2x_1x_0) = \overline{x}_5\overline{x}_4x_3\overline{x}_2\overline{x}_1x_0$$

described by the map (2), for $N = 64$ and $Q = 4$. In this instance, $a = x_5x_4, b = x_3x_2, c = x_1x_0$, and $\hat{a} = \overline{x}_5\overline{x}_4, \hat{b} = x_3\overline{x}_2, \hat{c} = \overline{x}_1x_0$. The sequences of permutations (5)–(8) is specified by the following maps.

$$x_5x_4, x_3x_2x_1x_0 \overset{\alpha_1}{\longmapsto} x_5x_4, x_3\overline{x}_2\overline{x}_1x_0$$
$$x_5x_4, x_3\overline{x}_2\overline{x}_1x_0 \overset{\alpha_2}{\longmapsto} \overline{x}_1x_0, x_3\overline{x}_2x_5x_4$$
$$\overline{x}_1x_0, x_3\overline{x}_2x_5x_4 \overset{\alpha_3}{\longmapsto} \overline{x}_1x_0, x_3\overline{x}_2\overline{x}_5\overline{x}_4$$
$$\overline{x}_1x_0, x_3\overline{x}_2\overline{x}_5\overline{x}_4 \overset{\alpha_4}{\longmapsto} \overline{x}_5\overline{x}_4, x_3\overline{x}_2\overline{x}_1x_0$$

Bandwidth, Area, and Time: The area and time performance of the folded BC-permutation network proposed above is a function of input size (N), wordlength (w), I/O-reduction factor (Q), and interconnection bandwidth r, where r is defined as the number of bits from each input element that can be read simultaneously ($1 \leq r \leq w$). Fig. 6 shows two transpose networks with $r = 1$ (1-BAAT design) and $r = 2$ (2-BAAT design). Both networks are designed for routing elements with word length $w = 6$. As stated in Fig. 6, circles represent storage cells with a multiplexed input, where the input to the left of the cell is selected during a horizontal-shift phase while the input from the bottom is

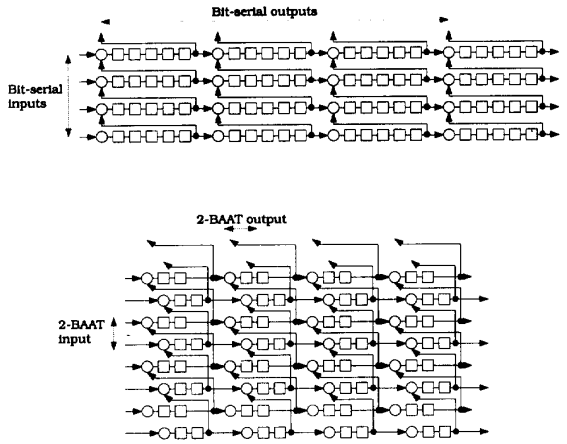


Fig. 6. 1–BAAT and 2–BAAT block-transpose networks for Q =4 and w =6. (Squares represent 1 b storage cells. Circles represent a 1 b storage cell with a multiplexed input.)

selected in a vertical-shift phase. The proposed networks perform transposition in two distinct phases. In the horizontal-shift phase, bits are shifted only horizontally until all bits have been read into the cells. Then, in the vertical-shift phase, data bits snake horizontally through square cells and vertically through circular cells until all bits have been output.

It is easy to show that an r-BAAT $Q \times Q$ transpose network with word length w, has area $O((w + r^2)Q^2)$ and time delay $O(wQ/r)$ units. Thus, the permutation network realizing α_2 or α_4 will have area $O((w + r^2)N)$ and time delay $O(wQ/r)$. For any $1 \leq r \leq w$, the interconnection networks realizing α_1 or α_3 are similar to the networks shown in Fig. 5 except that each line in the figure now indicates r wires routed in parallel. The area of these networks is thus $O(r^2N^2/Q^2)$, and their time delay is $O(wQ/r)$. Thus, the total delay of a folded BC-permutation network is $O(wQ/r)$, and its total area is $O(r^2N^2/Q^2 + (w + r^2)N)$. For optimal VLSI performance [3], [6], [13], [20], the range of Q must be restricted to $[1, \sqrt{N/w}]$, which leads to a "worst-case" area lower bound $A = \Omega((1 + r^2)wN)$, and $AT^2 = \Omega((1 + r^2/r^2)w^2N^2) = \Omega(w^2N^2)$.

V. Folded BP Permutation Networks

Using previous notation, let the string $R^q_{n-q}R^{n-q}_0$ denote an n-bit index, where R^q_{n-q} denotes the column number of the index and R^{n-q}_0 denotes the row number of the index. A permutation that routes P can be derived using a decomposition technique similar to that used in deriving BC-networks, however, with one additional complication: A BP permutation may involve an exchange (swap) of bits in R^q_{n-q} with bits in R^{n-q}_0. This shuffling of bits results in more complex permutations than just exchanging rows or columns of the input matrix, as was the case with BC permutations.

A BP permutation β can be defined by the following index-map:

$$x, y \xmapsto{\beta} u, v \tag{9}$$

where x, y denotes an n-bit input index such that $x = R^q_{n-q}$ specifies the column number and $y = R^{n-q}_0$ specifies the row number of an input element. Similarly, u, v denotes an n-bit output index such that $u = R^q_{n-q}$ specifies the row number and $v = R^{n-q}_0$ specifies the row number of an output element. The permutation β can be decomposed into a sequence of five permutations. However, to simplify description we will first decompose β into seven permutations $\beta_7 \cdots \beta_2\beta_1$, then show that some permutations can be combined.

Permutation β_1 is a *row-permutation* (i.e., a permutation involving row indices only) defined as follows:

$$x, y \xmapsto{\beta_1} x, cde \tag{10}$$

where $c = R^m_{n-q-m}, d = R^{n-2q-m}_q$, and $e = R^q_0$. Basically, β_1 is an index map that groups, into string c, those m bits ($m \leq q$) that must be moved from the row index (y) to the column index, as specified by the original permutation β. To be able to extract and permute bits from the column index x, the next permutation (β_2) swaps the indices x and e as follows:

$$x, cde \xmapsto{\beta_2} e, cdx. \tag{11}$$

Note that permutation β_2 is identical to permutation α_2 (or α_4) described in the previous section. The next three permutations ($\beta_{3'}, \beta_{3''}$, and $\beta_{3'''}$) extract and permute some bits from the index x, then form the final column index u, as follows:

$$e, cdx \xmapsto{\beta_{3'}} e, cdab \tag{12}$$

$$e, cdab \xmapsto{\beta_{3''}} e, bdac \tag{13}$$

$$e, bdac \xmapsto{\beta_{3'''}} e, bdu \tag{14}$$

where $b = R^m_0$ and $a = R^{q-m}_m$. Permutation $\beta_{3'}$ is an index map that groups, into string b, those m bits that must be moved from the column index (x) to the row index, as specified by β. Permutation $\beta_{3''}$ groups the strings a and c whose bits form the final column index after some permutation which is performed by permutation $\beta_{3'''}$. Since the previous three permutations are all row-permutations, they can be grouped into a single equivalent permutation $\beta_3 = \beta_{3'''}\beta_{3''}\beta_{3'}$. Two more permutations will complete the implementation of β, as follows:

$$e, bdu \xmapsto{\beta_4} u, bde \tag{15}$$

$$u, bde \xmapsto{\beta_5} u, v. \tag{16}$$

Permutation β_4 moves the final column index u to its proper position, and permutation β_5 permutes the bits of strings b, d, and e to yield the final row index v, as specified by the original permutation β. Thus, the permutation β can be decomposed into five permutations $\beta = \beta_5 \circ \beta_4 \circ \beta_3 \circ \beta_2 \circ \beta_1$. Note that the definitions of permutations β_2 and β_4 are identical to those of permutations α_2 and α_4 (Section IV). Thus, β_2 and β_4 can be routed using the same $Q \times Q$ transpose networks used for routing α_2 and α_4. The remaining permutations are row permutations which can be realized by direct interconnection. It is not difficult to show that the total delay of a folded BP-permutation network is $O(wQ/r)$, and its total area is $O(r^2N^2/Q^2 + (w + r^2)N)$. For useful operation, the range of

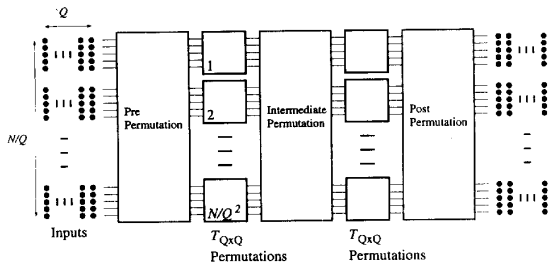


Fig. 7. General sturcture of folded BPC networks.

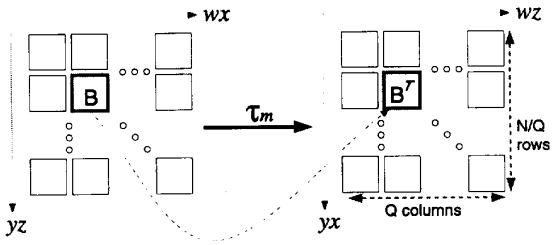


Fig. 8. Index-map of permutation $\tau_m(B$ is a $\frac{MQ}{N} \times \frac{MQ}{N}$ array).

Q must be restricted to $[1, \sqrt{N}]$, which leads to a total area of $O(wN + r^2N^2/Q^2)$.

As a final remark, it should be noted that a BC and a BP network can be merged into a single network with only two stages of folded transpose networks (rather than the four stages resulting from cascading a BC and a BP networks). First note that from the decompositions of permutations α and β, both permutations β_2 and α_2 can be routed using the same stage of $Q \times Q$ transpose networks, and permutations β_4 and α_4 can be routed also using the same stage of $Q \times Q$ transpose networks. The remaining row permutations can be composed pairwise (one from α and one from β). Specifically, a general BPC permutation γ can be expressed by the composition of five permutations $\gamma_5 \circ \gamma_4 \circ \gamma_3 \circ \gamma_2 \circ \gamma_1$, where

$$\begin{aligned}
\gamma_1 &= \beta_1 \circ \alpha_1 \quad \text{(pre permutation)} \\
\gamma_2 &= \beta_2 = \alpha_2 \quad \text{(block-transpose permutation)} \\
\gamma_3 &= \beta_3 \circ \alpha_3 \quad \text{(intermediate permutation)} \\
\gamma_4 &= \beta_4 = \alpha_4 \quad \text{(block-transpose permutation)} \\
\gamma_5 &= \beta_5 \quad \text{(post permutation)}.
\end{aligned}$$

Thus, for any BPC permutation, the proposed index-mapping methodology yields a five-stage permutation network. Two of these stages consist of a number of folded block-transpose networks. The remaining stages are simply a direct interconnection among the two stages of transpose networks and the I/O terminals. The general structure of folded BPC permutation networks is shown in Fig. 7. Note that to route any BPC permutations the two stages of block transpose networks remain fixed, only the pre, post, and intermediate stages of interconnection vary according to the specific permutation. provides the block-transpose networks can be prefabricated and connected by programmable interconnect stages.

VI. Precise Area Characterization

A more scrupulous look at the VLSI BPC networks derived above will reveal that their area is sensitive to position of the most significant bit affected by the given permutation. The worst-case area occurs when the BPC permutation involves the most significant bit of the indices. Otherwise, the area can be significantly smaller. For example, if the BPC permutation does not involve any of the q most significant bits (which form the column index), then transpose networks are not needed at all, and the total network area reduces to $O(N^2/Q^2)$. In general, if $x_m . n - 1 \geq m \geq 0$, is the most significant bit specified by some BPC permutation of size N, then this permutation can be realized by an r-BAAT network whose area $A_{\text{BPC}}(N, m, r)$ depends on N, m, and r in the following manner.

$$A_{\text{BPC}}(N, m) = \begin{cases} O(r^2N^2/Q^2 + (w + r^2)2^m) \\ \qquad \text{for } n - 1 \geq m \geq n - q \\ O\left(2^{(m+1)}\left(\frac{N}{Q}\right)\right) \\ \qquad \text{for } n - q - 1 \geq m \geq 0. \end{cases}$$

The reduction in VLSI area for the case $(n-q \leq m \leq n-1)$ is mainly due to the use of smaller transpose networks. Recall that the transpose networks were used to realize index-swap permutations $(\alpha_2, \alpha_4, \beta_2,$ and $\beta_4)$ which have the following form

$$a, bc \overset{\tau}{\longmapsto} c, ba$$

where, as before, a, bc denotes an n-bit index such that $a = R^q_{n-q}, b = R^{n-2q}_{n-q}$, and $c = R^q_0$. The above permutation can be reformulated to take advantage of the case $n-q \leq m \leq n-1$. The modified permutation (now denoted by τ_m) performs the following index mapping.

$$wx, yz \overset{\tau_m}{\longmapsto} wz, yx$$

where wx, yz denotes an n-bit index such that $w = R^{n-m}_m, x = R^{m-n+q}_{n-q}$, and $y = R^{2n-2q-m}_{m-n+q}, z = R^{m-n+q}_0$, and $n - q \leq m \leq n - 1$. The index map for this permutation is shown in Fig. 8.

The index map indicates that τ_m can be performed by partitioning the input data into blocks of size $(MQ/N) \times (MQ/N)$ then transposing each block, where $M = 2^m$. Consequently, the permutation network which routes τ_m consists of $(1/M)(N^2/Q^2)$ block-transpose networks each of size $(MQ/N) \times (MQ/N)$. For example, consider an 8×8 array of data (i.e., $N = 64$ and $Q = 8$) on which the following τ_4 permutation (i.e., $m = 4$) is to be performed:

$$x_5x_4x_3, x_2x_1x_0 \overset{\tau_4}{\longmapsto} x_5x_4x_0, x_2x_1x_3.$$

The permutation τ_4 has the following index map shown at the top of the next page.

For this example, $MQ/N = 2$ since $M = 2^m = 16, N = 64$ and $Q = 8$. Thus, the right hand array can be obtained by transposing each 2×2 block of the left hand array. The word-model permutation network which routes this permutation is shown in Fig. 9.

56	48	40	32	24	16	8	0		49	48	33	32	17	16	1	0
57	49	41	33	25	17	9	1		57	56	41	40	25	24	9	8
58	50	42	34	26	18	10	2		51	50	35	34	19	18	3	2
59	51	43	35	27	19	11	3	$\overset{\tau_4}{\longmapsto}$	59	58	43	42	27	26	11	10
60	52	44	36	28	20	12	4		53	52	37	36	21	20	5	4
61	53	45	37	29	21	13	5		61	60	45	44	29	28	13	12
62	54	46	38	30	22	14	6		55	54	39	38	23	22	7	6
63	55	47	39	31	23	15	7		63	62	47	46	31	30	15	14

56	48	40	32	24	16	8	0		35	34	33	32	3	2	1	0
57	49	41	33	25	17	9	1		43	42	41	40	11	10	9	8
58	50	42	34	26	18	10	2		51	50	49	48	19	18	17	16
59	51	43	35	27	19	11	3	$\overset{\tau_5}{\longmapsto}$	59	58	57	56	27	26	25	24
60	52	44	36	28	20	12	4		39	38	37	36	7	6	5	4
61	53	45	37	29	21	13	5		47	46	45	44	15	14	13	12
62	54	46	38	30	22	14	6		55	54	53	52	23	22	21	20
63	55	47	39	31	23	15	7		63	62	61	60	31	30	29	28

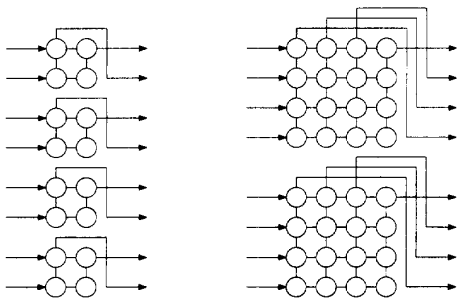


Fig. 9. Folded networks for τ_m for $m = 4$ and $m = 5$ $(N = 64, Q = 8)$.

A word-model network for routing the permutation

$$x_5x_4x_3, x_2x_1x_0 \overset{\tau_5}{\longmapsto} x_5x_1x_0, x_2x_4x_3$$

is also shown in Fig. 9 for the purpose of illustrating the increase in area when m is increased. Permutation τ_5 has the index map shown at the top of the page.

VII. Applications

The application of index mapping techniques in deriving folded permutation networks will be illustrated below by developing folded permutation networks for bit-reversal, digit-reversal, and K-shuffle permutations. These permutations are particular instances of BP permutations. Even though such permutations can be implemented by the general composition of BP permutations $\beta_5 \circ \beta_4 \circ \beta_3 \circ \beta_2 \circ \beta_1$, it will be shown below some permutations in the above sequence can be omitted for particular cases.

A. Bit and Digit Reversal Permutations

This type of permutations is encountered in computing Fast Fourier and other similar transforms. Bit-reversal is a BP-type permutation defined by the following mapping:

$$\rho(x_{n-1}x_{n-2}\cdots x_1x_0) = (x_0x_1\cdots x_{n-2}x_{n-1}). \tag{17}$$

To further simplify our presentation, the permutation ρ can be rewritten as

$$abc \overset{\rho}{\longmapsto} \check{c}\check{b}\check{a} \tag{18}$$

where $a = R^q_{n-q}, b = R^{n-2q}_q, c = R^q_0$, and $\check{a}$ denotes the bit-string obtained by bit-reversing string a; $\check{b}$ and $\check{c}$ have a similar connotation. Note that string a is the column index and bc is the row index of the input elements, while string $\check{c}$ is the column index and $\check{b}\check{a}$ is the row index of the final output elements. Permutation (18) can be decomposed into the following sequence of permutations.

$$abc \overset{\rho_1}{\longmapsto} ab\check{c} \tag{19}$$

$$ab\check{c} \overset{\rho_2}{\longmapsto} \check{c}ba \tag{20}$$

$$\check{c}ba \overset{\rho_3}{\longmapsto} \check{c}\check{b}\check{a}. \tag{21}$$

Note that permutation ρ_1 is equivalent to permutation β_1 (written, $\rho_1 \equiv \beta_1$) in the general decomposition of BP permutations (Section V). Also, $\rho_2 \equiv \beta_2$ and $\rho_3 \equiv \beta_3$. The complete structure of the reduced-area bit-reversal network is illustrated by the example of Fig. 10, for $N = 64$ and $Q = 4$.

Digit-reversal permutations are required for rearranging the inputs or outputs for multidimensional transforms with radix higher than 2, e.g., radix–4 Fast Fourier transform. A radix $-R$ $(R = 2^r)$ digit-reversal network has a structure similar to that of the folded bit-reversal network described above, especially when q is an integer multiple of r. In this case, the decomposition described by (19), (20) is still valid for

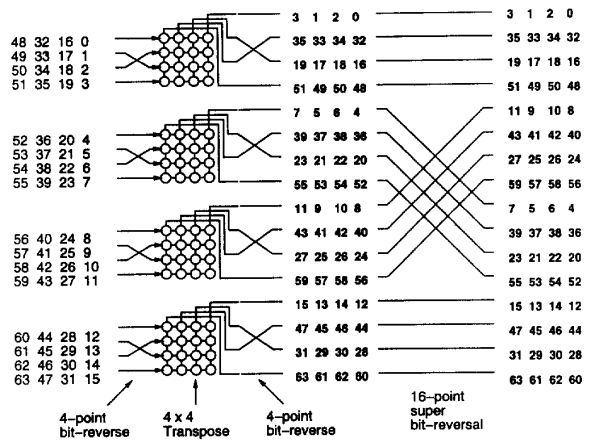


Fig. 10. A bit-reversal network for $N = 64$ and $Q = 4$.

digit-reversal. However, the definition of $\check{a}, \check{b}$, and $\check{c}$ must be modified provide radix$-R$ digit reversal. For example, if $c = x_5x_4x_3x_2x_1x_0$, and $r = 2$, then the radix–4 digit-reversed version of c is $\check{c} = x_1x_0x_3x_2x_5x_4$.

B. K-Shuffle Permutations

This is a fundamental class of permutations capable of realizing matrix transposition as well as index-rotation of multidimensional arrays. The class of *stride permutations* described in [5], [21] are shuffle permutations. K-shuffle permutations are a generalization of perfect shuffle permutations [18]. In the following, K-shuffle permutations will be specified in terms of $k = \log K, q = \log Q$, and $n = \log N$. Basically, two forms of folded K-shuffle networks can be derived depending on whether $K \leq Q$ or $K \geq Q$.

If $K \leq Q \leq \sqrt{N}$, then the K-shuffle permutation can be expressed by the following index-map:

$$ab, cde \overset{\sigma}{\longmapsto} bc, dea \tag{22}$$

where $a = R^k_{n-k}, b = R^{q-k}_{n-q}, c = R^k_{n-q-k}, d = R^{n-2q-k}_q$, and $e = R^q_0$. Applying the technique of Section V, the permutation σ can be decomposed into the following sequence of permutations:

$$ab, cde \overset{\sigma_1}{\longmapsto} e, cdab \tag{23}$$

$$e, cdab \overset{\sigma_1}{\longmapsto} e, adbc \tag{24}$$

$$e, adbc \overset{\sigma_3}{\longmapsto} bc, ade \tag{25}$$

$$bc, ade \overset{\sigma_4}{\longmapsto} bc, dea. \tag{26}$$

Careful scrutiny of the above permutations reveals that $\sigma_1 \equiv \beta_2, \sigma_2 \equiv \beta_3, \sigma_3 \equiv \beta_4$, and $\sigma_4 \equiv \beta_5$, where the permutations $\beta_2, \beta_3, \beta_4, \beta_5$ are as defined in Section V. A folded permutation network that implements σ is shown in Fig. 11 for $N = 64, Q = 4$, and $K = 2$.

When $K \geq Q$, the K-shuffle permutation can be expressed by the following index-map.

$$a, bcd \overset{\varsigma}{\longmapsto} c, dab \tag{27}$$

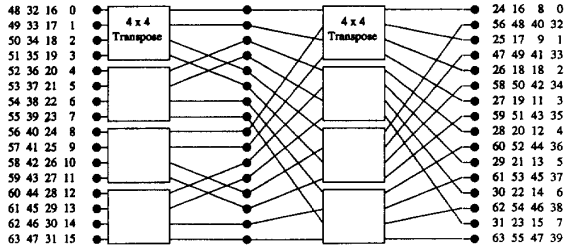


Fig. 11. A folded K–shuffle network for $N = 64, Q = 4, K = 2$.

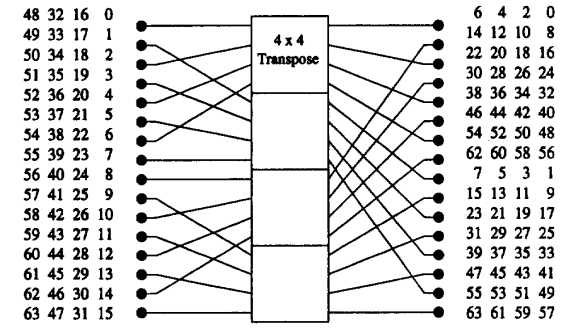


Fig. 12. A folded K–shuffle network for $N = 64, Q = 4, K = 8$.

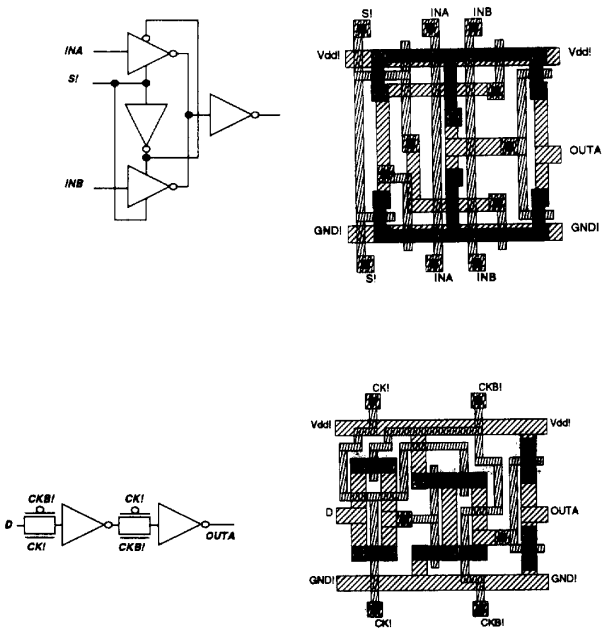


Fig. 13. Circuits and layouts of the two basic components: A multiplexer and a 1 b shift-register cell.

where $a = R^q_{n-q}, b = R^{k-q}_{n-k}, c = R^q_{n-q-k}$, and $d = R^{n-q-k}_0$. The permutation ς can be decomposed into the following sequence of permutations:

$$a, bcd \overset{\varsigma_1}{\longmapsto} a, bdc \tag{28}$$

$$a, bdc \overset{\varsigma_2}{\longmapsto} c, bda \tag{29}$$

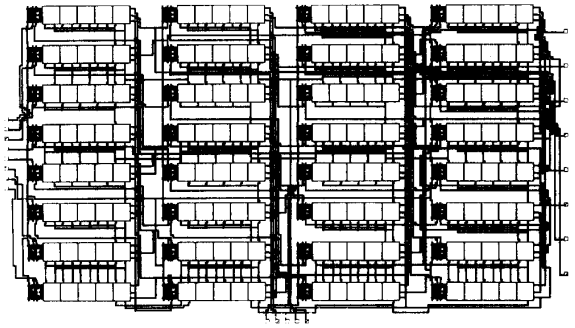


Fig. 14. Layout of a 4 × 4, 2–BAAT block-transpose network showing the placement of register cells (for elements of length $w = 10$ b).

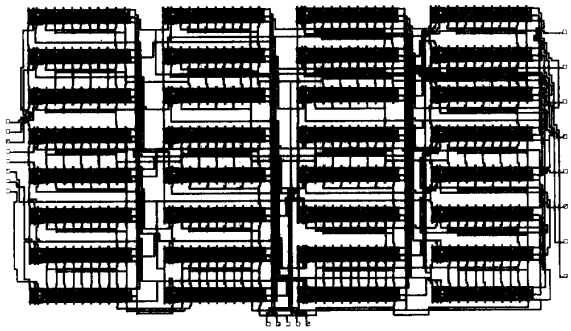


Fig. 15. Layout of a 4 × 4, 2–BAAT block-transpose network showing the internal layout of register cells.

$$c, bda \xrightarrow{\varsigma_3} c, dab. \tag{30}$$

It easy to notice that $\varsigma_1 \equiv \beta_1, \varsigma_2 \equiv \beta_2$, and $\varsigma_4 \equiv \beta_{543}$. A folded network that implements permutation ς is shown in Fig. 12 for $N = 64, Q = 4$, and $K = 8$.

VIII. CMOS VLSI Layouts

K-shuffle permutations form an important subclass of the permutations in the BPC class, because they are capable of transposing matrices or rotating the index-space of multidimensional arrays in one routing step. These networks have been used in computing multidimensional transforms [1], [5], [6], [9], sorting [3], [13], and a large number of other applications. To demonstrate the efficiency of our mapping techniques, a VLSI layout has been generated and simulated for the folded 8-shuffle network of Fig. 12. The layouts have been generated using the MAGIC environment [14] And the placement of cells was carried out using an enhanced version of the CFL (coordinate free lap) library routines [7]. Functional correctness was verified using logic simulators and timing analysis was carried out using HSPICE. and simulation was carried used verilog as well as HSPICE. Fig. 13 shows the custom layouts of the two basic cells used in Fig. 6. A CMOS VLSI layout for a 4 × 4, 2-BAAT, block-transpose network for elements of word length $w = 10$ b is shown at two levels of detail in Figs. 14 and 15. The layout has dimensions $1854\lambda \times 1097\lambda$, where $\lambda = 0.6$ microns for a 1.2-micron technology. The layout of the entire 64-point folded 8-shuffle network is shown in Fig. 16. The dimensions of the entire layout is $4165\lambda \times 2098\lambda$ (about 2.5 mm × 1.26 mm), resulting in a total VLSI area (without I/O pads) of 3.15 mm^2.The functional correctness of the layout (without including I/O pads) has been verified at internal clock speeds exceeding 100 MHz. It should be pointed out that the 1.2-micron process used for the above layout is not state-of-the-art. The area can be reduced significantly and the clock speed can be increased by using state-of-the-art technologies.

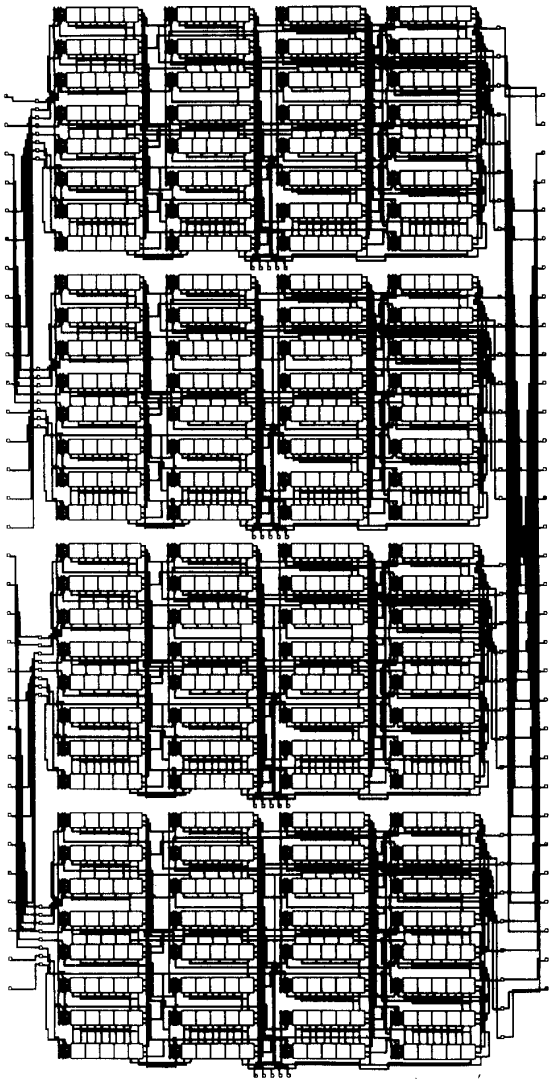


Fig. 16. Complete layout of a 64-point 8–shuffle network with $Q = 4$.

IX. Conclusions

Interconnection network folding is a very useful tool for mapping networks with a large number of inputs and outputs onto VLSI architectures with limited I/O. This paper presented a systematic approach for designing folded networks for routing the BPC class of permutations. The main attribute of this

approach is that any folded BPC network can be constructed from smaller block-transpose networks. The derived networks have uniform and regular structure resulting in very compact VLSI layouts that can route permutations at very high clock speeds. Furthermore, the folded-network hardware can be made reprogrammable by using programmable interconnect to realize the pre, post, and intermediate permutations of Fig. 7. This type of interconnect programmability is readily achievable by the antifuse technology used with field programmable gate arrays [10], and other available technologies.

The basic work of this paper can be extended in a number of directions. One direction is generalizing the index-mapping technique to realize other classes of permutations. Another direction, is to incorporate other design parameters in the folding technique. The techniques proposed in this paper basically involve two design parameters, the number of inputs (N) and the I/O reduction factor (Q), where chip area and delay are optimized with respect to these parameters. It is conceivable that other parameters can be incorporated in the folding process, such as layout aspect-ratio, power dissipation, maximum wire-length, etc. In this case, however, the optimization procedure will be considerably more complex.

REFERENCES

[1] H. M. Alnuweiri, "Optimal VLSI networks for multidimensional transforms," *IEEE Trans. Parallel, Distrib. Syst.*, vol. 5, pp. 763–769, July 1994.

[2] ——, "Routing BPC permutations in VLSI," in *Proc. 6th Int. Parallel Process. Symp.*, 1992, pp. 116–119.

[3] ——, "A new class of optimal bounded-degree VLSI sorting networks," *IEEE Trans. Comput.*, vol. 42, June 1993, pp. 746–752.

[4] H. M. Alnuweiri, S. M. Sait, and M. Darwish, "Efficient routing of a class of permutations in VLSI," in *Proc. Brown/MIT Conf. Adv. Res. VLSI*, 1992, pp. 348–365.

[5] M. An, I. Gertner, M. Rofheart, and R. Tolimieri, "Discrete fast fourier transform algorithms: A tutorial survey," *Adv. Electron., Electron Physics*, vol. 80, pp. 1–67, 1991.

[6] G. Bilardi, S. W. Hornick, and M. Sarrafzadeh, "Optimal VLSI architectures for multidimensional DFT," in *ACM Symp. Parallel Algorith. Architect.*, Santa Fe, NM, June 1989, pp. 365–272.

[7] W. Beckett, "CFL—Coordinate free lap," *Reference manual (Version 1.2)*, Northwest LIS Rel. 3.1, Univ. Washington, 1987.

[8] P. Duris, O. Sykora, I. Vrt'o, and C. D. Thompson, "Tight chip area lower bounds for discrete Fourier and Walsh-Hadamard transformations," *Inform. Process. Lett.*, vol. 21, pp. 245–247, 1985.

[9] A. Elnaggar, H. M. Alnuweiri, and M. R. Ito, "Mapping tensor products onto VLSI networks with reduced I/O," submitted for publication.

[10] J. Greene, E. Hamdy, and S. Beal, "Antifuse field programmable gate arrays," *Proc. IEEE*, vol. 81, pp. 1042–1056, 1993.

[11] D. Kleitman, F. T. Leighton, M. Lepley, and G. L. Miller, "New layouts for the shuffle-exchange graph," in *Proc. ACM Symp. Theory Comput.*, 1981, pp. 278–292.

[12] F. T. Leighton, "New lower bound techniques for VLSI," *Mathematical Syst. Theory*, vol. 17, pp. 47–80, 1984.

[13] ——, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. Comput.*, vol. C-34, pp. 344–354, Apr. 1985.

[14] R. N. Mayo, "1990 DECWRL/Livermore MAGIC release," Univ. California, Berkeley, Sept. 1990.

[15] D. Nassimi and S. Sahni, "Optimal BPC permutations on a cube connected SIMD computer," *IEEE Trans. Comput.*, vol. C-31, pp. 338–341, 1982.

[16] F. Preparata and J. Vuillemin, "Area-time optimal VLSI networks for multiplying matrices," *Inform. Proc. Lett.*, vol. 11, no. 2, Oct. 1990.

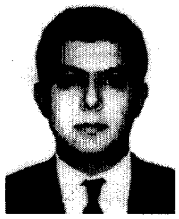
[17] A. R. Seigel, "Minimum storage sorting circuits," *IEEE Trans. Comput.*, vol. C-34, pp. 355–361, 1985.

[18] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.

[19] C. D. Thompson, "Area-time complexity for VLSI," in *Proc. ACM Symp. Theory Comput.*, May 1979.

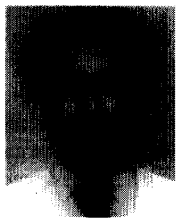
[20] J. Ullman, *Computational Aspects of VLSI*. Rockville, Md: Comput. Sci. Press, 1984.

[21] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*. Reading, PA: SIAM, 1992.



Hussein M. Alnuweiri (S'89–M'89) received the B.S. and M.S. degrees in 1983 and 1984, respectively, both in electrical engineering from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, and the Ph.D. degree also in electrical engineering from the University of Southern California, Los Angeles in 1989 .

From 1989 to 1991 he was with the Computer Engineering Department at King Fahd University of Petroleum and Minerals. Currently, he is an Assistant Professor in the Department of Electrical Engineering at the University of British Columbia. He was the recipient of a 1992 New Faculty Award from the British Columbia Advanced Systems Institute. His research interests include computer architecture, parallel and distributed computing, highly parallel VLSI structures, reconfigurable interconnection networks, and asynchronous transfer mode (ATM) switch design and optimization.



Sadiq M. Sait received the B.E. degree in electronics from the Bangalore University in 1981, and the M.S, and Ph.D. degrees in electrical engineering from King Fahd University of Petroleum and Minerals (KFUPM) in 1983 and 1986, respectively.

He is currently an Associate Professor in the Department of Computer Engineering. He is the principal author of the book *VLSI Physical Design Automation: Theory and Practice*, Europe: McGraw-Hill Book Co. He has authored or coauthored over 50 papers in international journals and conferences, and has contributed two chapters to the book *Progress in VLSI Design*. He served on the editorial board of the *International Journal of Computer Aided Design* between 1988–1990, and was an invited guest editor for their special issue on Hardware Description Languages. His current interests are digital design automation, VLSI system design, and high-level synthesis.

In 1981 Dr. Sait received the Best Electronics Engineer Award from the IEE, Bangalore. In 1990 and 1994, he was awarded the Excellence in Research Award by KFUPM.