# Programming Cellular Permutation Networks Through Decomposition of Symmetric Groups

A. YAVUZ ORUÇ, MEMBER, IEEE, AND M. YAMAN ORUÇ

*Abstract*—**A fundamental problem in interconnection network theory is to design permutation networks with as few cells as possible and a small programming or setup time. The well-known networks of Benes and Waksman have asymptotically optimal cell counts, but the best setup algorithm available for such networks with $n$ inputs requires $O(n \log_2 n)$ sequential time. As an alternative, this paper considers another class of permutation networks which are collectively referred to as cellular permutation arrays. Using a group theoretic formulation, a natural correspondence is established between such permutation networks and iterative decompositions of symmetric groups through cosets. Based on this correspondence, the setup problem is reduced to iteratively determining the leaders of the cosets to which the permutation to be realized belongs. This, in turn, leads to linear-time setup algorithms for cellular permutation arrays. The paper describes these algorithms in detail for two families of cellular permutation arrays reported in the literature.**

*Index Terms*—**Cellular permutation array, control algorithm, coset, group decomposition, permutation group, permutation network, symmetric group.**

## I. INTRODUCTION

DURING the last two decades, the researchers in interconnection network theory have concentrated their efforts on $n$-input networks with $O(n \log_2 n)$ binary switches or cells, in view of the fact that such structures are optimal with respect to the number of their switches [2]–[4], [8], [9], [14], [20]–[26]. As a consequence of these efforts, asymptotically optimal networks, such as Benes–Waksman–Joel constructions were discovered [3], [8], [24], and other $O(n \log_2 n)$ networks, such as baseline, omega, and cube networks, which can be made rearrangeable by feedback connections have been proposed [4], [10], [18], [20], [25], [26]. The discovery of asymptotically optimal permutation networks is a celebrated result in interconnection network theory. However, as profound as this discovery may be, it has also led to an interesting paradox. The recent research results [11], [13], [17]–[19], [25], [26] appear to conjecture that determining the states of the switches of these networks to realize a specified permutation requires at least order of $n \log_2 n$ sequential time which contrasts with $O(\log_2 n)$ data propagation delay in such networks. Clearly, this mismatch makes overlapping the data propagation with the decoding of permutations very difficult and overall setup time remains asymptotic to $n \log_2 n$. As a remedy, the setup time can be brought down to $O(\log_2^2 n)$ by using an $n$-processor computer whose processors are interconnected by a complete graph [13]. However, it seems cost ineffective to use an $n$-processor computer which requires $O(n^2)$ connections to setup a network with $O(n \log_2 n)$ switches. Our remark also remains valid for other parallel setup algorithms suggested in [11], [13].

The foregoing discussion indicates that optimizing the performance of a permutation network over the number of its switches alone is not necessarily the best strategy, and that one must also consider the control aspect of the problem. With this as a basis, we consider another class of rearrangeable networks which are collectively referred to as cellular permutation arrays [1], [5], [6], [12], [15], [16], [21]–[23]. Despite the fact that these networks may require up to $O(n^2)$ switches, they have several attractive features. First, cellularity implies local connections and no criss-crossings between cells as in $O(n \log_2 n)$ networks. All interconnections are confined into cells which can easily be implemented. Second, as we demonstrate in the paper, an $n$-input cellular permutation array can be setup or programmed in $O(n)$ sequential time. This result combined with the fact that such a network has also $O(n)$ propagation delay leads to the design of a simple control unit which overlaps data propagation with permutation decoding efficiently. Consequently, the total delay for realizing arbitrary permutations remains asymptotic to $n$. Finally, the cellular permutation arrays considered in this paper have a triangular geometry which makes them very attractive for VLSI implementation since the data path and control unit can compactly be placed on a rectangular grid as described in [12], [17].

The paper is organized as follows. Section II provides the algebraic notation and background which form the basis for the setup algorithms presented in subsequent sections. More specifically, permutations groups, cosets, and iterative decompositions of symmetric groups through cosets are described. In Sections III and IV further characterizations of these decompositions via specific maps such as transpositions and cycles are given. It is shown that there is a natural correspondence between iterative coset decompositions of symmetric groups and cellular permutation arrays. In Sections V and VI, this correspondence is utilized to develop linear time algorithms for

Manuscript received July 5, 1985; revised July 28, 1986. This work was supported in part by the New York State Government under Grant C000821.

A. Y. Oruc is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180.

M. Y. Oruc is with the Department of Mathematics, Syracuse University, Syracuse, NY 12180.

IEEE Log Number 8714101.

programming cellular permutation arrays. The design of a control unit for decoding specified permutations is given in Section VII and the paper is concluded in Section VIII.

## II. Basic Concepts

A permutation network is a triplet $(S, D, P)$ where $S(D)$ is a set of symbols called input(output) terminals, and $P$ is a set of one-to-one maps from $S$ onto $D$. The sets $S$ and $D$ may represent physically disjoint sets of input and output terminals as in two-sided networks such as telephone networks or connectors between processor elements and memory units in array processors, or they may correspond to the same set of terminals as in one-sided permutation networks such as those that interconnect processor elements in array processors. Since we shall primarily be concerned with the maps from $S$ to $D$ rather than the actual physical locations that $S$ and $D$ represent, we identify the terminals by integers $1, 2, \cdots, n$, and let $S = D = \{1, 2, \cdots, n\}$. Accordingly, the elements of set $P$ will be viewed as permutations over $\{1, 2, \cdots, n\}$.

Let $i, j \in S$ and $p \in P$. We shall write $(i)p = j$ to imply that $i$ is moved to $j$ under map $p$. Furthermore, we shall compose maps from left to right so that for permutations $p$ and $q$ over $S$, $p \cdot q$ is defined as $((s)p)q$ for all $s \in S$. Both cycle and two-row matrix forms will be used to represent permutations [7]. Thus, we write $p = (s_1 s_2 \cdots s_r)$ for some $s_1, s_2, \cdots, s_r \in S$; $2 \le r \le n$ to imply that $(s_i)p = s_{i+1}$; $1 \le i \le r - 1$ and $(s_r)p = s_1$, and represent the same map in two-row matrix form as

$$p = \begin{pmatrix} s_1 s_2 \cdots s_r & s_{r+1} s_{r+2} \cdots s_n \\ s_2 s_3 \cdots s_1 & s_{r+1} s_{r+2} \cdots s_n \end{pmatrix}$$

where $s_i \in S - \{s_1, s_2, \cdots, s_r\}$; $r + 1 \le i \le n$. It is known that any permutation can be expressed as a product of disjoint cycles, and that this product is unique, except for the order of its cycles.

The set of all permutations over $S$ forms a group, called the symmetric group of degree $n$, and denoted by $\Sigma_n$ or $\Sigma_S$ to emphasize the set $S$ where $|\Sigma_n| = n!$. A permutation network is called a *permuter* or a *rearrangeable network* if its set of permutations is all of $\Sigma_n$. A rearrangeable network can be constructed in several ways [1]–[4], [8], [9], [14], [20]–[26]. The $n$-input permuters considered in this paper are constructed by decomposing the symmetric group $\Sigma_i$ into the cosets of $\Sigma_{i-1}$ in $\Sigma_i$; $i = 2, 3, \cdots, n$ [15].

Let $(G,.)$ be a group and $(H,.)$ be a subgroup of $G$. The set of elements $h.g$ for all $h \in H$ and fixed $g \in G$ is called a *right coset* of $H$ in $G$, and denoted $Hg$. Similarly, the set of elements $g \cdot h$ for all $h \in H$ and fixed $g \in G$ is called a *left coset* of $H$ in $G$, and denoted $gH$. The number of right (left) cosets of $H$ in $G$ is called the index of $H$ in $G$. The following results from group theory [7] form the basis for the decompositions of $\Sigma_n$ which we shall use throughout the paper.

*Proposition 1:* Two right (left) cosets of $H$ in $G$ are either disjoint or identical. ‖

*Proposition 2:* The number of elements in a right (left) coset of $H$ in $G$ is equal to the number of elements in $H$. Furthermore, the number of elements in $G$ is equal to the product of the index of $H$ in $G$ and the number of elements of $H$. ‖

We shall write $G = He + Hg_2 + Hg_3 + \cdots + Hg_k$ where $e$ is the identity element of $G$ and $k$ is the index of $H$ in $G$, to imply that the right cosets $H, Hg_2, \cdots, Hg_k$ are pairwise disjoint, and that they span all of $G$. The elements $e, g_2, g_3, \cdots, g_k$ are called the *right coset leaders* of $H$ in $G$. Similarly, we shall write $G = eH + g_2'H + g_3'H + \cdots + g_k'H$ for left cosets of $H$ in $G$, and call $e, g_2', g_3', \cdots, g_k'$ the *left coset leaders* of $H$ in $G$. Note that $h \cdot g_i$ for any $h \in H$ is also a right coset leader of $Hg_i$; $1 \le i \le k$ since $H(h.g_i) = Hg_i$. Similarly, $g_i' \cdot h$ for any $h \in H$ is a left coset of $g_i'H$; $1 \le i \le k$ since $(g_i' \cdot h)H = g_i'H$.

It is seen that cosets form a basis for decomposing any finite group into a collection of disjoint sets of elements. This decomposition is unique up to the subgroup $H$ and its coset leaders in $G$. These facts were used in [15] to transform symmetric groups into rearrangeable permutation arrays. Here, we shall consider two decompositions of $\Sigma_n$ which are easily transformable to cellular permutation arrays and are corollaries of the following general observation.

*Theorem 1:* Let $p_1, p_2, \cdots, p_n \in \Sigma_n$.

$$\Sigma_n = \Sigma_{n-1}p_1 + \Sigma_{n-1}p_2 + \cdots + \Sigma_{n-1}p_n \tag{1}$$

if and only if $(n)p_i \ne (n)p_j$ for all $i, j$; $1 \le i \ne j \le n$, and

$$\Sigma_n = p_1\Sigma_{n-1} + p_2\Sigma_{n-1} + \cdots + p_n\Sigma_{n-1} \tag{2}$$

if and only if $(n)p_i^{-1} \ne (n)p_j^{-1}$ for all $i, j$; $1 \le i \ne j \le n$.

*Proof:* Consider the first part of the statement. Observe that every permutation in $\Sigma_{n-1}$ maps $n$ to $n$, and hence every permutation in $\Sigma_{n-1}p_i$ must map $n$ to $(n)p_i$ and that in $\Sigma_{n-1}p_j$ must map $n$ to $(n)p_j$. But $(n)p_i \ne (n)p_j$ by hypothesis, and hence $\Sigma_{n-1}p_i \cap \Sigma_{n-1}p_j = \Phi$. Furthermore, since this is true for all $i, j$; $1 \le i \ne j \le n$, the sufficiency follows. To prove the necessity, suppose for some $i, j$; $1 \le i \ne j \le n$, $\Sigma_{n-1}p_i \cap \Sigma_{n-1}p_j = \Phi$, and $(n)p_i = (n)p_j$. Then $(n)p_i \cdot p_j^{-1} = ((n)p_i) \cdot p_j^{-1} = ((n)p_j) \cdot p_j^{-1} = n$, and hence $p_i \cdot p_j^{-1} \in \Sigma_{n-1}$ or $p_i \in \Sigma_{n-1}p_j$. But this contradicts the assumption that $\Sigma_{n-1}p_i \cap \Sigma_{n-1}p_j = \Phi$ since $p_i \in \Sigma_{n-1}p_i$. We conclude that $(n)p_i \ne (n)p_j$ for all $i, j$; $1 \le i \ne j \le n$. The second part of the statement is argued similarly. ‖

Intuitively, the rationale behind the above theorem is that the permutations in $\Sigma_n$ are partitioned to the cosets of $\Sigma_{n-1}$ according to where the symbol $n$ is mapped to in the case of right cosets and where it is mapped from in the case of left cosets. Those permutations that map $n$ to $n$ form $\Sigma_{n-1}$, those that map $n$ to 1 form $\Sigma_{n-1}p_1$ where $(n)p_1 = 1$, those that map $n$ to 2 form $\Sigma_{n-1}p_2$ where $(n)p_2 = 2$ and so on. The only constraint on maps $p_1, p_2, \cdots, p_n$ is that they map $n$ to different symbols. Similarly, in the case of left cosets, those permutations that map $n$ form $\Sigma_{n-1}$, those that map 1 to $n$ form $p_1\Sigma_{n-1}$, where $(1)p_1 = n$, those that map 2 to $n$ form $p_2\Sigma_{n-1}$ where $(2)p_2 = n$, and so on. This idea is pictured for right cosets more concretely in Fig. 1.

The decompositions that are given in the next two sections are based on these observations and results.

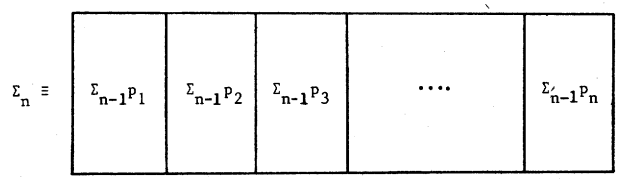



Fig. 1. Decomposition of $\Sigma_n$ into the right cosets of $\Sigma_{n-1}$; $p_1 = e$; $(n)p_i = i - 1$; $2 \leq i \leq n$.

## III. Decomposition with Transpositions

Consider the symmetric groups $\Sigma_2, \Sigma_3, \cdots, \Sigma_n$. It follows from Theorem 1 that

$$\begin{aligned}
&\Sigma_n = \Sigma_{n-1}e + \Sigma_{n-1}(1\ n) + \Sigma_{n-1}(2\ n) + \cdots + \Sigma_{n-1}(n-1\ n)\\
&\Sigma_{n-1} = \Sigma_{n-2}e + \Sigma_{n-2}(1\ n-1) + \Sigma_{n-2}(2\ n-1)\\
&\qquad + \cdots + \Sigma_{n-2}(n-2\ n-1)\\
&\vdots\\
&\Sigma_3 = \Sigma_2 e + \Sigma_2(13) + \Sigma_2(23)\\
&\Sigma_2 = \{e, (12)\}.
\end{aligned} \tag{3}$$

Similarly,

$$\begin{aligned}
&\Sigma_n = e\Sigma_{n-1} + (1\ n)\Sigma_{n-1} + (2\ n)\Sigma_{n-1} + \cdots + (n-1\ n)\Sigma_{n-1}\\
&\Sigma_{n-1} = e\Sigma_{n-2} + (1\ n-1)\Sigma_{n-2} + (2\ n-1)\Sigma_{n-2}\\
&\qquad + \cdots + (n-2\ n-1)\Sigma_{n-2}\\
&\vdots\\
&\Sigma_3 = e\Sigma_2 + (13)\Sigma_2 + (23)\Sigma_2\\
&\Sigma_2 = \{e, (12)\}.
\end{aligned} \tag{4}$$

Equations (3) and (4) are the iterative decompositions of $\Sigma_n$ into the right and left cosets of $\Sigma_{n-1}, \Sigma_{n-2}, \cdots, \Sigma_2$, respectively. In both cases, the identity permutation, $e$ and transpositions $(i\ j)$; $1 \leq i \leq j - 1$ are the coset leaders of $\Sigma_{j-1}$ in $\Sigma_j$; $3 \leq j \leq n$. These two decompositions are the group theoretic representations of what are known as regular and reverse triangular permutation arrays with binary switching cells [15]. In order to see this relation in concrete terms, the triangular arrays for $n = 5$ are depicted in Fig. 2. In accordance with the terminology introduced in [15], [17], we shall refer to these two networks as KLW and reverse KLW networks.

Let us further consider the 5-input KLW network depicted in Fig. 2(a). It is seen that the cell in the left most column can be set to either of the maps $e$ and (12), and hence the first column realizes $\Sigma_2$. Moreover, we can set the cells in the second column from the left to permutations $e$, (13), (23) and (13)(23). Of these four maps, the first three are the right coset leaders of $\Sigma_2$ in $\Sigma_3$. Consequently, the first two columns of cells realize $\Sigma_3 = \Sigma_2 e + \Sigma_2(13) + \Sigma_2(23)$. Note that the map (13)(23) = (123) is not needed in the realization. Thus, $\Sigma_3$ is realized by setting at most one cell of the second column to a transposition map at a given time. The same observations can

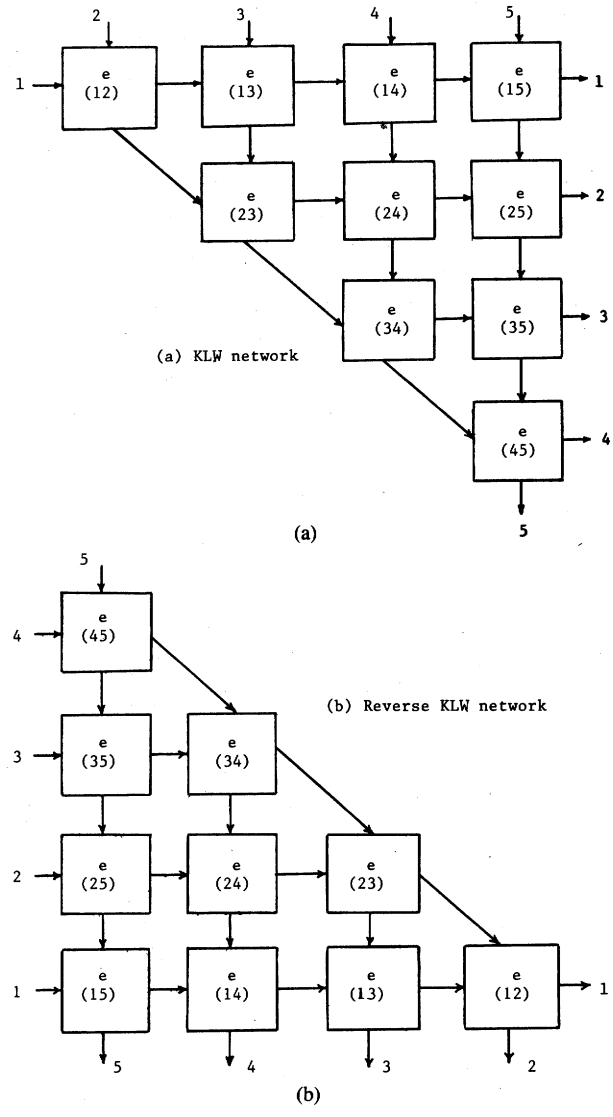


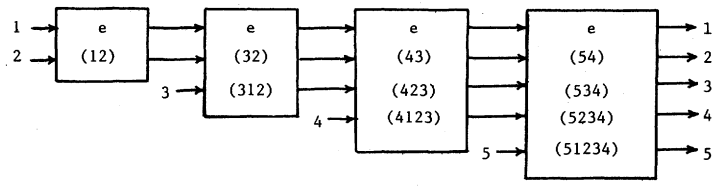

Fig. 2. 5-input triangular permutation networks.

be made about the third and fourth columns from which we conclude that the entire network must realize $\Sigma_5$. The relation between $n$-input reverse KLW network and the left coset decomposition given in (4) is demonstrated similarly.

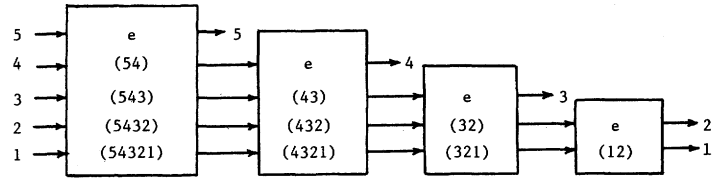
## IV. Decomposition with Cycles

It is also possible to decompose $\Sigma_n$ iteratively into the cosets of $\Sigma_i$; $2 \leq i \leq n - 1$ by using cycle maps as coset leaders. Using Theorem 1, it can be shown that

$$\begin{aligned}
&\Sigma_n = \Sigma_{n-1}e + \Sigma_{n-1}(n\ n-1) + \Sigma_{n-1}(n\ n-2\ n-1)\\
&\qquad + \cdots + \Sigma_{n-1}(n\ 1\ 2\ \cdots\ n-1)\\
&\Sigma_{n-1} = \Sigma_{n-2}e + \Sigma_{n-2}(n-1\ n-2) + \Sigma_{n-2}(n-1\ n-3\ n-2)\\
&\qquad + \cdots + \Sigma_{n-2}(n-1\ 1\ 2\ \cdots\ n-2)\\
&\vdots\\
&\Sigma_3 = \Sigma_2 e + \Sigma_2(32) + \Sigma_2(312)\\
&\Sigma_2 = \{e, (12)\}.
\end{aligned} \tag{5}$$




(a) BBC network




(b) Reverse BBC network

Fig. 3. 5-input cascade permutation networks.

Similarly,

$$\Sigma_n = e\Sigma_{n-1} + (n\ n-1)\Sigma_{n-1} + (n\ n-1\ n-2)\Sigma_{n-1} + \cdots + (n\ n-1\ \cdots\ 2\ 1)\Sigma_{n-1}$$

$$\Sigma_{n-1} = e\Sigma_{n-2} + (n-1\ n-2)\Sigma_{n-2} + (n-1\ n-2\ n-3)\Sigma_{n-2} + \cdots + (n-1\ n-2\ \cdots\ 2\ 1)\Sigma_{n-2}$$

$$\vdots \qquad \vdots \qquad (6)$$

$$\Sigma_3 = e\Sigma_2 + (32)\Sigma_2 + (321)\Sigma_2$$

$$\Sigma_2 = \{e,\ (12)\}.$$

The above decompositions are the group-theoretic representations of what are known as BBC and reverse BBC networks, respectively [15]. As an example, we depict the 5-input BBC and reverse BBC networks in Fig. 3. The cycle maps shown inside the cells with three or more inputs are coset leaders, while the cells with two inputs realize $\Sigma_2$. Although the connections inside the cells are not explicitly shown, cycles define these connections completely. For example, the cycle (5234) in the rightmost cell in Fig. 3(a) implies that inputs 5, 2, 3, and 4 are connected to outputs 2, 3, 4, and 5, respectively, and input 1 is connected to output 1. The entire cells is designed by establishing the connections dictated by all the permutations it must perform.

Note that the total number of permutations in both $n$-input BBC and KLW networks is $n(n-1)/2$ and equal to the number of coset leaders that appear in the iterative decompositions (3)–(6). The main difference between the two realizations is in how many permutations are packed into each cell. In a KLW network each cell contains exactly one coset leader while in a BBC network, cells may contain up to $n$ permutations. This fact accounts for the difference between the numbers of cells in the two realizations. In fact, one can distribute coset leaders over cells in a variety of ways. The interested reader is referred to [15] for other alternatives.

## V. Setup Procedure for KLW Networks

In this section we shall describe procedures to set up KLW networks for arbitrary permutations. Recall from (3) that the

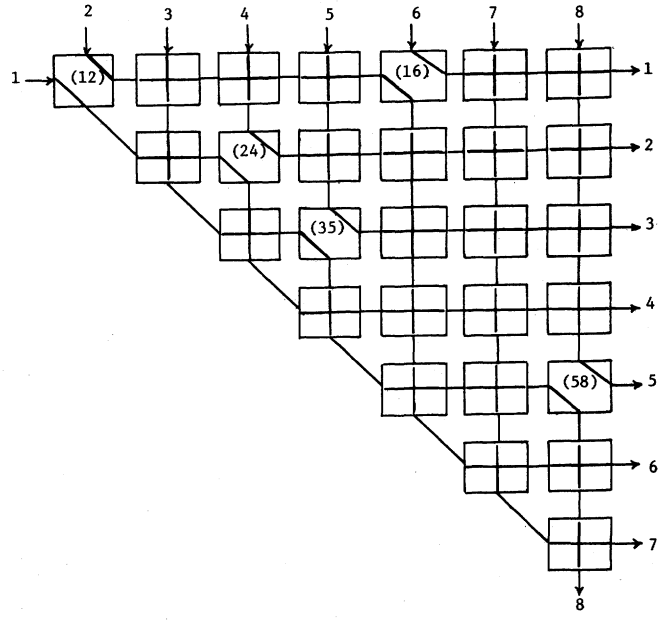



Fig. 4. 8-input KLW network shown to realize $p = (1426)(385)$.

$j$th column of $n$-input KLW network is programmable for each of the coset leaders in the set $Q_j = \{e, (1\ j+1), (2\ j+1), \cdots, (j\ j+1)\}$ where the $i$th transposition $(i\ j+1)$ is located in the $i$th row and $j$th column of the network. With this characterization, each map of $n$-input KLW network can be expressed as a product $q_1q_2 \cdots q_{n-1}$ for some $q_j \in Q_j$; $1 \le j \le n-1$. As an example, for the 8-input KLW network shown in Fig. 4, if we let $q_1 = (12)$, $q_2 = e$, $q_3 = (24)$, $q_4 = (35)$, $q_5 = (16)$, $q_6 = e$, and $q_7 = (58)$, the network realizes the map $q_1 \cdot q_2 \cdots q_7 = (12)(24)(35)(16)(58) = (1426)(385)$. Based on this observation, Oruc and Prakash [16] developed an algorithm for programming KLW networks for arbitrary permutations. The serial time complexity of that algorithm was shown to be in the order of $n \log_2 n$ for $n$-input KLW network. Here we present an algorithm whose serial time complexity is linear in $n$.

First consider an example. Let $n = 16$ and $p = (14)(2\ 7\ 9)(3\ 6\ 5\ 8)$. Start with symbol 9, and write $p = p_1 \cdot (9\ (9)p)$. Determine $p_1$ from this expression as $p_1 = p \cdot (9\ 2) =$

(14)(27)(3658). Note that $9(p_1) = 9$. Consequently, $p$ is decomposed into two permutations one of which is a coset leader which belongs to a cell in the last column of 9-input KLW network, and another which is a permutation in $\Sigma_8$, and hence should be realizable by the first seven columns of that network. Another way to state this is that $p = p_1.(9\ 2)$ is located in the right coset $\Sigma_8(9\ 2)$ with $p_1 \in \Sigma_8$. Now repeat the same process for $p_1$ and symbol 8 by letting $p_1 = p_2.(8\ (8)p_1) = p_2.(8\ 3)$. Determine $p_2$ from this expression as $p_2 = (14)(27)(365)$. Now, notice that $p_1$ is decomposed into $p_2.(83)$ where $p_2 \in \Sigma_7$. As a result $p$ is decomposed into $p_2.(83)(92)$ with $p_2 \in \Sigma_7$. Thus, if this decomposition process is repeated for $p_2$, $p_3$, and so on up to $p_7 \in \Sigma_2$, we obtain $p = ee(41)(53)(65)(72)(83)(92)$. The map $p$ is then realized by setting the cells of 9-input KLW network according to the coset leaders that appear in the decomposition.

The steps described above can be stated formally as in the following recursive procedure, wherein $i$ is initialized to $n$ before the procedure starts, and $f$ denotes the decomposed form of $p$.

```
Procedure KLWFACTOR (p, i; var f, q);
If i=1
    then
        f:=e
            else begin
                q:=(i (i)p);
                p:=p.(i (i)p);
                f:=KLWFACTOR (p,i-1).q;
                endbegin;
endif;
endprocedure;
```

We note that the above procedure differs from the algorithm given in [16] in two respects. First, unlike that algorithm it requires no sorting, and furthermore it works with the entire permutation instead of its cycles. The symbol which specifies the coset leader for a column is determined according to the fixed sequence $n, n-1, \cdots, 1$, once $i$ is initialized to $n$. Hence, no sorting is necessary. Consequently, the procedure terminates in $O(n)$ steps where each step involves a composition, i.e., $p := p.(i\ (i)p)$; $1 \le i \le n$ and, a call to the procedure. Everytime KLWFACTOR is called, the map $p$ is factored into a coset leader and a new permutation $p := p(i\ (i)p)$ which is known to be in the right coset $\Sigma_{i-1}.(i\ (i)p)$. This process repeats itself until $i = 1$. We remark that it is possible to tighten KLWFACTOR by avoiding the composition $p.(i(i)p)$ when it is not necessary. There are two cases for which this composition becomes redundant; when $(i)p = i$ and, when $i$ appears in a transposition of $p$. Avoiding the composition for these two cases can improve the running time of KLWFACTOR considerably and can easily be inserted into the procedure by simple if statements.

Procedure KLWFACTOR can be modified to program $n$-input reverse KLW network. Simply replace in that procedure $q := (i(i)p)$ by $q := (i(i)p^{-1})$, $p := p \cdot q$ by $p := q \cdot p$, and $f :=$ KLWFACTOR$(p, i-1) \cdot q$ by $f := q \cdot$KLWFACTOR $(p, i-1)$. The reader may easily verify that $(i\ (i)p^{-1})$ is the appropriate left coset leader for decomposing $p$ into the cosets of $\Sigma_{i-1}$ in $\Sigma_i$; $3 \le i \le n$ and that the modified procedure also takes $O(n)$ steps to run.

## VI. Setup Procedure for BBC Networks

The setup procedure given in the preceding section can be extended to programming BBC and reverse BBC networks. Recall from (5) that the $i$th stage of $n$-input BBC network is programmable for each of the coset leaders in the set

$$P_i = \{e, (i+1\ i), (i+1\ i-1\ i), \cdots, (i+1\ 1\ 2\ 3 \cdots i)\};$$
$$1 \le i \le n-1.$$

Unlike the cells of a KLW network, each cell of a BBC network is programmable for the identity and a number of cycle maps over its terminals. Thus, instead of decomposing a specified permutation $p$ into a product of transpositions, we must factor it into an appropriate product of cycles which are coset leaders of $\Sigma_{i-1}$ in $\Sigma_i$; $3 \le i \le n$.

As an example, let $n = 6$ and $p = (1325)(46)$. We factor $p$ into the coset leaders of $\Sigma_{i-1}$ in $\Sigma_i$; $3 \le i \le 6$ as follows. Start with symbol 6 and write $p = p_1 \cdot (6\ (6)p \cdots 4\ 5)$ or $p = p_1 \cdot (645)$. Determine $p_1$ from this expression as $p_1 = p \cdot (645)^{-1} = (13245)$ and note that $(6)p_1 = (6)p \cdot (645)^{-1} = 6$. Consequently, $p = p_1 \cdot (645)$ is located in the right coset $\Sigma_5 \cdot (645)$ of $\Sigma_6$ with $p_1 \in \Sigma_5$. Now repeat the same process for $p_1$ and symbol 5. That is, let $p_1 = p_2 \cdot (5\ (5)p \cdots 34)$ or $p_1 = p_2 \cdot (51234)$ and solve $p_2$ from this expression as $p_2 = p_1 \cdot (51234)^{-1} = (123)$. As a result $p$ is determined to be in the coset $\Sigma_4(51234)(645)$ with $p_2 \in \Sigma_4$. Arguing similarly, it is, in general, shown that if $p_i = p_{i+1} \cdot (n-i(n-i)p_i \cdots n-i\ -3\ n-i\ -2\ n-i-1)$; $0 \le i \le n-2$ then $n-i$ remains fixed under $p_{i+1}$. Thus, if the above process is repeated three more times we factor $p$ as $p = e(312)e(51234)(645)$. The map $p$ is then realized by setting the cells of 6-input BBC network according to the coset leaders in the decomposition.

The steps of the above example can be generalized into the following recursive procedure wherein $f$ denotes the decomposed form of $p$, and $i$ is initialized to $n$ before the procedure starts.

```
Procedure BBCFACTOR (p, i; var f, q);
If i=1
    then
        f:=e
            else begin
                q:=(i (i)p···i-3 i-2 i-1)
                p:=p·(i(i)p···i-3 i-2 i-1)^-1
                f:=BBCFACTOR(p,i-1)·q
                endbegin;
endif;
endprocedure;
```

Procedure BBCFACTOR can be modified to program $n$-input reverse BBC network by replacing $q := (i\ (i)p \cdots i-3\ i-2\ i-1)$ by $q := (i\ i-1\ i-2 \cdots (i)p^{-1})$, $p := p \cdot (i\ (i)p \cdots i-3\ i\ -2\ i-1)^{-1}$ by $p := (i\ i-1\ i-2 \cdots (i)p^{-1})^{-1} \cdot p$ and $f :=$ BBCFACTOR $(p, i-1) \cdot q$ by $f := q \cdot$ BBCFACTOR $(p, i-1)$. The reader may verify that both

BBCFACTOR and its modified version terminate in $O(n)$ steps.

## VII. Permutation Decoding and Control

The setup procedures presented in the preceding sections can be implemented by a simple control unit. In order to provide a detailed description but otherwise without loss of generality, we shall discuss the design of a control unit for a reverse KLW network. There are two reasons for this choice. First, there is a straightforward, one-to-one correspondence between the addresses and maps generated by the control unit of such a network and the coordinates of the cells to be programmed. Second, when factorizing a permutation into the coset leaders residing in the cells of a reverse KLW network, we begin with the leftmost column where the data enter the permuter and proceed to the right one column at a time until the last column. As a result, the data and factorization advances in the same direction. This situation allows for overlapping the factorization with data propagation, and thus eliminates the extra setup time which would otherwise be necessary. It can easily be shown that it takes an $n$-input cellular permutation array $n$ steps to complete its operation with overlapping and $2n$ steps without overlapping. We remark that reverse BBC networks behave like reverse KLW networks and are also suitable for overlapping. On the other hand, neither KLW nor BBC networks possess this overlapping property since in both cases, the data and permutation factorization proceed in opposite directions.

The control unit for a reverse KLW network is organized as shown in Fig. 5. It consists of a factorizer and two decimal decoders. The factorizer iteratively decomposes the specified permutation into a product of coset leaders which are transpositions realizable by the network. Whenever a transposition is factored out, the $R$ and $C$ inputs of the row and column decoders, respectively, are fed with the symbols which appear in that transposition. Consequently, the appropriate decoder outputs activate the column and the row at the intersection of which the cell containing that transposition is located.

In order to see how the factorization is performed in detail, recall from the modified version of KLWFACTOR that the specified permutation $p$ is factored by successive compositions $p = (i\,(i)p^{-1}).p$; $i = n, n-1, \cdots, 2$. Let $p_{\text{old}}$ and $p_{\text{new}}$ denote the map $p$ before and after the composition, respectively. It is easily verified that $(i)p_{\text{new}} = i$; $((i)p_{\text{old}}^{-1})p_{\text{new}} = (i)p_{\text{old}}$, and $(j)p_{\text{new}} = (j)p_{\text{old}}$ for all $j$; $1 \le j \le n$; $j \ne i$ and $j \ne (i)p_{\text{old}}^{-1}$. Now note that once $(i\,(i)p_{\text{old}}^{-1})$ is factored out, we let $C := i$ and $R := (i)p_{\text{old}}^{-1}$ and the column of cells with the $i$th output is set to $(i\,(i)p_{\text{old}}^{-1})$. After that, we are only interested in the images of $i-1, i-2, \cdots, 1$ under the map $p_{\text{new}}$. Hence, the fact that $(i)p_{\text{new}} = i$ is of no consequence. Moreover, since $(j)p_{\text{new}} = (j)p_{\text{old}}$ for all $j$; $1 \le j \le n$ except for $j = i$ and $j = (i)p_{\text{old}}^{-1}$, the only mapping that must be performed to obtain $p_{\text{new}}$ from $p_{\text{old}}$ is $((i)p_{\text{old}}^{-1})p_{\text{new}} = (i)p_{\text{old}}$. This just amounts to replacing the old image of $(i)p_{\text{old}}^{-1}$ by $(i)p_{\text{old}}$.

In order to perform the above operations, the factorizer is equipped with a dual memory system as shown in Fig. 6.

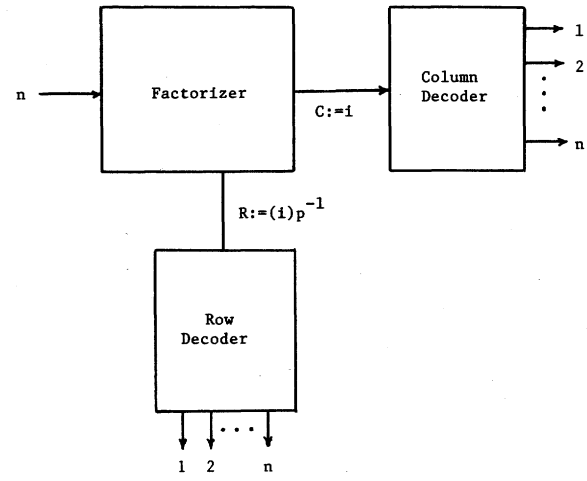



Fig. 5. The organization of control unit for reverse KLW network.

Suppose that the specified permutation $p$ and its inverse $p^{-1}$, respectively, reside in memories denoted by $P$ and $Q$ with the associated address and data registers also depicted in the figure. The factorizer executes the following sequence of operations for $i = n, n-1, \cdots, 2$:

1) Factor $(i\,(i)p^{-1})$

$$C := i;\ R := Q[i]$$

2) Update $p$ and $p^{-1}$

$$P[Q[i]] := P[i];\ Q[P[i]] := Q[i].$$

Step 1) of the above sequence can easily be implemented by loading $i$ into the address register QADR and reading $Q[i]$ into $R$ through QDTR. The second one requires the following subsequence of operations:

(2.1) PADR := $i$; QADR := $i$

(2.2) PDTR := $P$[PADR]; QDTR := $Q$[QADR]

(2.3) PADR := QDTR; QADR := PDTR

(2.4) $P$[PADR] := PDTR; $Q$[QADR] := QDTR

An example may clarify the steps described above. Let $p = \left(\begin{smallmatrix}1&2&3&4&5&6\\4&3&6&2&5&1\end{smallmatrix}\right)$. The following snapshots show the formation of the factorization $p = (63)(41)(32)(21)$ as well as the changes in the contents of memories $P$ and $Q$ as $i$ runs through 6, 5, $\cdots$, 2.

| Step: 1 2 3 4 5 | Step: 1 2 3 4 5 | Step: 1 2 3 4 5 |
|---|---|---|
| $i$= 6 5 4 3 2 | 4 4 4 2 2 | 6 3 3 3 2 |
| $C$= 6 5 4 3 2 | 3 3 3 3 1 | 4 4 4 1 1 |
| $R$= 3 5 1 2 1 | $P$= 6 1 1 1 X | $Q$= 2 2 2 2 X |
| | 2 2 2 X X | 1 1 1 X X |
| | 5 5 X X X | 5 5 X X X |
| | 1 X X X X | 3 X X X X. |

Note that the values of $C$ and $R$ are specified according to the column and row numbering scheme depicted in Fig. 1. Also note that X's in $P$ and $Q$ correspond to mappings $(i)p = i$ and

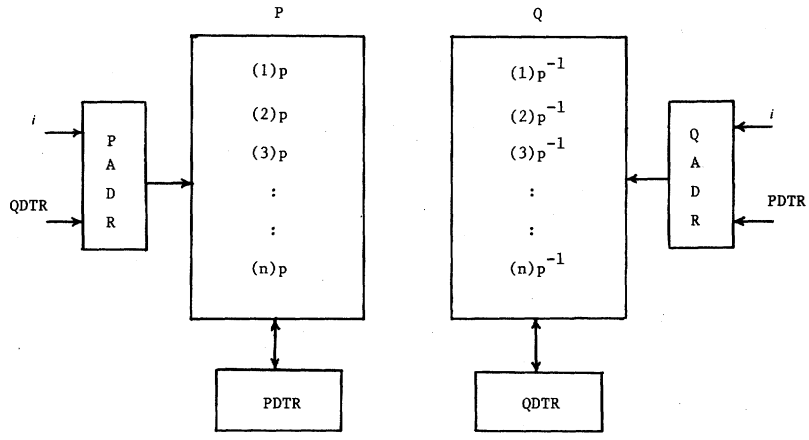



Fig. 6. Memory organization for the factorizer.

$(i)p^{-1} = i$; $i = 6, 5, \cdots, 2$. Finally notice that, when $(i\ (i)p^{-1}) = (i\ i)$, i.e., when the coset leader for column $i$ is identity, the decoder outputs specify that the cells located at $C = i$ and $R = i$ be set to the coset leader $(i\ (i)p^{-1})$. However, there is no such cell in column $i$, and it is obvious that, in this case, all the cells in column $i$ are to be set to the identity. Thus, when $i \leq (i)p^{-1}$, the decoder outputs are overridden by a flag which indicates that all the cells in column $i$ should be set to the identity. These and other details concerning the internal logic design of cells are explored further in [17].

## VIII. Summary and Concluding Remarks

The paper has presented a new approach for permutation network design and control using decompositions of symmetric groups into cosets. It has been shown that there is a natural and direct relation between such decompositions and cellular permutation arrays. This has been demonstrated for two families of cellular permutation arrays which appeared in [1] and [9]. Based on these results, linear-time algorithms have been given to set up these families of networks. As a result, it has been shown that the $O(n \log_2 n)$ setup time problem of optimal permutation networks can be overcome by employing cellular permutation arrays that contain redundant cells.

In the networks considered here, this redundancy exhibits itself either directly in terms of cell counts as in KLW networks or in terms of the total number of connections over all the cells as in BBC networks. In both cases, the total number of coset leaders and hence connections is $O(n(n-1)/2)$ for $n$ inputs while the number of cells in KLW networks is $n(n-1)/2$ and that for BBC networks is $n-1$. In general, the total number of coset leaders is fixed by the decomposition of $\Sigma_n$ and cannot be altered without modifying the subgroup over which the coset decomposition is based. The cellular permutation networks considered here are based on the decomposition given in Theorem 1, and hence they all consist of $n(n-1)/2$ coset leaders. However, these can be distributed over permutation cells in a variety of ways [15]. If, on the other hand, permutation networks with fewer coset leaders are desired then different subgroups must be used in decompositions of symmetric groups. This possibility is currently being investigated.

Another aspect of the decomposition approach which has not been discussed in the paper is its application to fault-tolerant interconnection network design. Cosets form a natural basis to partition the elements of $\Sigma_n$ into disjoint sets, and hence may be used to distinguish among valid and faulty permutations. The fault tolerance of a permutation network may thus be thought of as its ability to remain within the coset that contains the valid permutation despite faults. Further elaboration of this subject and other applications of the coset decomposition approach will be deferred to another place.

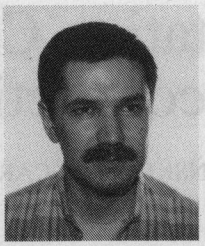
## Acknowledgment

The authors thank anonymous referees for their valuable comments and criticisms.

## References

[1] S. Bandyopadhyay, S. Basu, and K. Choudhury, "A cellular permuter array," *IEEE Trans. Comput.*, vol. C-21, pp. 1116–1119, Oct. 1972.
[2] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic.* New York: Academic, 1965.
[3] ——, "Permutation groups, complexes, and rearrangeable connecting networks," *Bell Syst. Tech. J.*, vol. 43, pp. 1619–1640, July 1964.
[4] T. Feng, "A survey of interconnection networks," *Computer,* vol. 14, pp. 12–27, Dec. 1981.
[5] J. Gecsei, "Interconnection networks from three-state cells," *IEEE Trans. Comput.*, vol. C-26, pp. 705–711, Aug. 1977.
[6] J. Gecsei and J.-P. Brassard, "The topology of cellular partitioning networks," *IEEE Trans. Comput.*, vol. C-30, pp. 164–168, Feb. 1981.
[7] M. Hall, Jr., *The Theory of Groups.* New York: Chelsea, 1976.
[8] A. E. Joel, "On permutation switching networks," *Bell Syst. Tech. J.*, pp. 813–822, May–June 1968.
[9] W. H. Kautz, K. N. Levitt, and A. Waksman, "Cellular interconnection arrays," *IEEE Trans. Comput.*, vol. C-17, pp. 443–451, May 1968.
[10] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-30, pp. 324–332, May 1981.
[11] G. F. Lev, N. Pippenger, and L. G. Valiant, "A fast parallel algorithm for routing in permutation networks," *IEEE Trans. Comput.*, vol. C-30, pp. 93–100, Feb. 1981.
[12] S. A. Nadkarni, "The design and performance evaluation of hybrid cellular interconnection arrays," M.Sc. Thesis, Rensselaer Polytechnic Instit., Troy, NY, Aug. 1985.
[13] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Benes permutation network," *IEEE Trans. Comput.*, vol. C-31, pp. 148–154, Feb. 1982.
[14] D. C. Opferman and N. T. Tsao-Wu, "On a class of rearrangeable switching networks," *Bell Syst. Tech. J.*, pp. 1579–1618, May–June 1971.

[15] Y. A. Oruc, "Symmetric groups, cosets and cellular permutation arrays," in *Proc. 20th Inform. Sci. Syst. Conf.*, Princeton, NJ, 1986.
[16] Y. A. Oruc and D. Prakash, "Routing algorithms for cellular interconnection arrays," *IEEE Trans. Comput.*, vol. C-33, pp. 769–762, Aug. 1984.
[17] Y. A. Oruc, "Designing cellular permutation arrays through decomposition of symmetric groups into cosets," *J. Parallel Distributed Comput.*, to be published.
[18] D. S. Parker, Jr., "Notes on shuffle/exchange-type switching networks," *IEEE Trans. Comput.*, vol. C-29, pp. 213–222, Mar. 1980.
[19] ——, "New points of view on three-stage rearrangeable switching networks," in *Proc. Workshop Interconnection Networks,* IEEE Comp. Soc., 1980, pp. 56–63.
[20] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.
[21] K. J. Thurber, "Permutation switching networks," in *Proc. 1971 Comput. Designers' Conf.*, Jan. 1971. pp. 7–24.
[22] ——, "Interconnection networks—A survey and assessment," in *Proc. Nat. Comput. Conf.*, 1974, pp. 909–919.
[23] ——, "Circuit switching technology: A state of the art survey," in *Proc. COMPCON,* Fall 1978, Sept. 1978, pp. 116–124.
[24] W. A. Waksman, "A permutation network," *J. Ass. Comput. Mach.*, pp. 159–163, May 1968.
[25] C. W. Wu and T. Feng, "The reverse-exchange interconnection network," *IEEE Trans. Comput.*, vol. C-29, pp. 801–811, Sept. 1980.
[26] ——, "The university of shuffle-exchange network," *IEEE Trans. Comput.*, vol. C-30, pp. 324–332, May 1981.



**A. Yavuz Oruç** (S'81–M'83) received the B.Sc. degree in electrical engineering from Middle East Technical University, Ankara, Turkey in 1976, the M.Sc. degree in electronics from the University of Wales, Cardiff, U.K., in 1978, and the Ph.D. degree in electrical engineering from Syracuse University, Syracuse, NY, in 1983.

Since January 1983, he has been on the faculty of the Department of Electrical Computer and Systems Engineering at Rensselaer Polytechnic Institute, Troy, NY. His research interests include parallel processing, and analysis and design of connection and computation networks for advanced computer systems.

Dr. Oruc is a member of the IEEE Computer Society.

**M. Yaman Oruç** received the B.Sc. degree in mathematics from Middle East Technical University, Ankara, Turkey in 1976, and the M.Sc. degree in mathematics from Syracuse University, Syracuse, NY in 1982.

She is currently completing the requirements for the Ph.D. degree in mathematics also at Syracuse University. Her research interests include algebraic topology, in particular, equivariant topology, and applied mathematics.

Ms. Oruc is a member of American Mathematical Society and Sigma Xi.