Design and Analysis of a Generalized Architecture for Reconfigurable m-ary Tree Structures

Sampalli Srinivas, *Member, IEEE*, and Nripendra N. Biswas, *Senior Member, IEEE*

Abstract— **This paper presents a generalized architecture for reconfigurable m-ary tree structures, where m is any integer > 1. The approach is based on a generalized multistage interconnection network (MIN) which is a generalization of the augmented shuffle-exchange MIN introduced by the authors in [1] for obtaining reconfigurable binary tree structures. The generalized architecture with m^k processing elements or nodes (where k is any integer > 1) is implemented with a k- stage MIN. A single control code issued to the MIN establishes a distinct m-ary tree configuration among the nodes. The architecture can assume $m.2^{\lceil \log_2 m \rceil (k-1)}$ distinct m-ary tree configurations, one for each value of the control code. The favorable features of the architecture include fast reconfiguration, simplified hardware in the nodes and the MIN, and simple routing control.**

The generation of reconfigurable m-ary tree structures is based on a generalized reconfiguration equation. Analysis of this equation is carried out to prove the reconfigurability of the architecture. The results of the analysis are utilized to provide a procedure to synthesize the m-ary tree configuration that is generated for any given control code. Furthermore, considerations for implementing the switching elements of the MIN are discussed.

Index Terms— **Intra-topology reconfiguration, m-ary shuffle, m-ary tree structures, multistage interconnection networks, reconfigurable parallel architecture, switching elements, switching theory.**

I. Introduction

RECONFIGURATION of hardware resources in parallel architectures has been advocated as an attractive design strategy to implement high-performance systems. The motivations behind this approach to parallel processing are to speed up algorithm execution, minimize inter-processor communication delays, improve resource utilization, and provide for fault-tolerance by restoring the system's connectivity on the occurrence of faults. Several reconfigurable parallel systems of different philosophy, reconfiguration technique, and application scope have been proposed [2]–[28].

In general, a parallel architecture consisting of a network of processors is characterized by the network topology or *structure*, and is modeled as a graph, with the nodes representing the processors and the edges representing the communication links. The *configuration* of a parallel architecture refers to its structure in which the location of each node has been specified. A *reconfigurable parallel architecture* is one whose configuration can be altered between different algorithm executions or between phases of the same algorithm execution. Typically, such an architecture should be capable of three types of reconfiguration [29]. The first is *intra-topology reconfiguration*, which enables the architecture to assume different configurations corresponding to the same structure. This will enhance the inter-processor communication efficiency and the fault-tolerance capability of the architecture. The second is *inter-topology reconfiguration*, which refers to altering the topology of the architecture so as to establish a network structure that is well-mapped to the algorithm structure, thereby achieving higher computational efficiency. The third is *partitioning*, by which the architecture can operate as a number of independent configurations, thus enhancing resource utilization. Previously proposed parallel architectures with one or more of the above types of reconfiguration include the Kartashevs' dynamic architecture [2], Star [3], CHiP [4], GF 11 [5], Connection machine [6], Pradhan's network architecture [7], RCAP [8], and architectures equipped with reconfigurable bus systems [9]–[11]. Furthermore, a number of reconfigurable architectures have been proposed purely to support fault-tolerance by the utilization of spares [12]–[24]. It should be also mentioned here that in some other types of reconfigurable parallel systems such as PASM [25], TRAC [26], Opsila [27], and IBM RP3 [28], reconfiguration refers mainly to the capability of obtaining a mixed-mode parallelism (SIMD and MIMD or shared memory and local memory modes) and partitionability in such modes.

Of the various network topologies that have been proposed, the tree structure has received much interest as a versatile architecture for parallel processing [30]–[47]. Many tree-structured architectures have been proposed for various applications including those based on divide-and-conquer algorithms, searching, sorting, and other dictionary operations [31]–[36], numeric and combinatorial problems [37], parallel processing of large databases [38], [39], production systems [40], and control algorithms [41]. Some tree-structured parallel architectures that have been built are the Dado machine [40], Aspen [42], Non-Von [43], and Teradata's DBC [44]. These architectures are mainly *static* in nature, that is, they do not have the reconfiguration capability. In order to enhance the fault-tolerance and routing efficiency, tree-structured architectures augmented with additional static links have been proposed (e.g., [31], [45]–[47]).

Manuscript received April 26, 1988; revised April 28, 1992.

S. Srinivas was with the Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore - 560012, India. He is now with the Division of Computing Science, Department of Mathematics, Statistics, and Computing Science, Dalhousie University, Halifax, NS, B3H 3J5 Canada.

N. N. Biswas was with the Center for Advanced Computer Studies, University of Southwestern Louisiana, LA 70504. He is now with the Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore-560012, India.

IEEE Log Number 9202609.

The performance of a tree-structured parallel architecture can be significantly enhanced if it is equipped with the capability of intra-topology reconfiguration. Such a reconfigurable architecture can assume a number of distinct tree configurations. As mentioned earlier, this type of reconfiguration enhances the inter-processor communication efficiency and the fault-tolerance capability of the architecture. For instance, this capability may be utilized to bring two data-exchanging nodes adjacent to each other, thereby reducing the communication delays. Furthermore, by this capability, the faulty nodes can be isolated by purging them into the leaf positions, so that the system's connectivity is not lost and it continues to operate in a gracefully degraded mode.

This paper is concerned with the design of a generalized architecture for obtaining intra-topology reconfiguration in m-ary tree structures, where m is any integer > 1. The proposed architecture for the reconfigurable m-ary tree structure generates distinct m-ary tree configurations (TC's), where each TC is an m-ary tree structure.

Some of the previously proposed reconfigurable architectures have the capability of intra-topology reconfiguration in binary tree structures. In the Kartashevs' dynamic architecture [2], the processors are interconnected by a multiple-bus based crossbar type network. A distinct binary tree configuration is obtained for each value of a control code issued to a shift-register with variable bias (SRVB) placed inside each node. Other topologies such as stars and multirooted trees can be obtained by a structural variation in the SRVB. The Star system [3] is another architecture which is able to configure a pool of processors into a tree which can be rooted at any arbitrary processor. The system is based on a baseline multistage interconnection network (MIN). Reconfiguration is obtained by executing algorithms that set the switches of the network so that the processors are configured as a binary tree structure. Other approaches for configuring binary tree topologies with multistage networks include the GF 11 [5] and RCAP [8]. The CHiP computer [4] is a single-chip architecture capable of reconfiguring into different binary tree configurations with prestored commands. Pradhan's network architectures [7] are able to assume binary tree configurations through logical reconfiguration, that is, in the absence of physical switching. In addition to these, a number of reconfigurable binary tree architectures have been proposed purely for fault-tolerance by the utilization of spares [12]–[24].

Any reconfiguration scheme introduces two types of overhead: the reconfiguration hardware, which is the extra hardware support required to perform reconfiguration, and reconfiguration time, which is the delay in switching between configurations. One of the primary considerations in designing a reconfigurable system is to minimize the overheads introduced by reconfiguration, in terms of both time and hardware cost.

The authors of this paper recently reported a novel architecture for reconfigurable binary tree structures satisfying the above objectives [1]. The reconfigurable binary tree architecture with $N = 2^k$ nodes is capable of assuming N distinct binary tree configurations (BTC's). The architecture is implemented with a k-stage augmented shuffle-exchange MIN. The existing shuffle-exchange MIN [48] does not have the capability of assuming a BTC among the nodes. This capability has been imparted to the network by a simple augmentation. The reconfiguration in the architecture is based on the following reconfiguration equation. Suppose that each node is denoted as $P(i)$, $i = 0, 1, \cdots, N-1$. The equation gives the relation between each node $P(i)$ and the node $P(j)$ to which it gets connected for each value of a k-bit control code C controlling the switching in the MIN:

$$P(j) = (CRS[P(i)] \Lambda B) \oplus C \tag{1}$$

where $CRS[P(i)]$ is a one-bit circular right shift of the binary representation of $P(i)$, B is a k-bit number with value $b_{k-1}b_{k-2}\ldots b_1 b_0 = 11\ldots10$, Λ is the bitwise AND, and $\oplus$ is the bitwise EXOR operation. It has been proved in [1] that an architecture which establishes interconnections among its nodes according this equation is able to assume distinct BTC's.

This paper presents a generalized architecture for reconfigurable m-ary tree structures, where m is any integer > 1. The design is based on the generalization of the reconfigurable binary tree architecture. The nodes in the architecture are interconnected by a generalized MIN. The reconfigurability of the architecture is based on a generalized reconfiguration equation, which is again a generalization of (1).

The architecture for a reconfigurable m-ary tree structure with $N = m^k$ nodes (where k is any integer > 1) is implemented with a k-stage MIN. A single control code issued to the MIN achieves reconfiguration. A distinct m-ary TC is established among the nodes for each value of the control code. The architecture can assume $m.2^{\lceil \log_2 m \rceil (k-1)}$ distinct m-ary TC's.[1] Note that if m is a power of 2, the number of distinct configurations is m^k. The favorable features of the architecture include fast reconfiguration, simplified hardware in the nodes and the MIN, and simple routing control.

An analysis of the generalized reconfiguration equation is carried out, and the proof for reconfiguration is given based on the analysis. The results obtained while proving for reconfiguration are utilized to provide a procedure to synthesize the m-ary tree configuration from the root node for any given control code. Design considerations for efficient implementation of the switching elements of the MIN are also discussed.

As mentioned above, the proposed approach can be used to design a reconfigurable m-ary tree architecture, for any integral value of $m > 1$. The principles of the approach have their basis in the design of the architecture when m is a power of 2. There are a few basic modifications in the design strategy when m is an integer which is not a power of 2. We present the design of the architecture for the former case and then outline the differences in the design when m is not a power of 2.

The rest of the paper is organized as follows. Section II gives the preliminaries that are useful in understanding the proposed reconfigurable m-ary tree structure and the architecture. Section III describes the design of the generalized architecture for the case when m is a power of 2. The generalized reconfiguration equation is stated and analyzed,

[1] $\lceil x \rceil$ represents the smallest integer greater than or equal to x.

and the proof for reconfiguration is given based on the analysis. The results obtained while proving for reconfiguration are utilized to provide a procedure to synthesize the m-ary TC from the root node. Section IV outlines the differences in the design of the architecture for the case when m is not a power of 2. Section V discusses some issues in the design and implementation of the switching elements of the MIN. Section VI draws concluding remarks and enumerates the advantages of the proposed architecture.

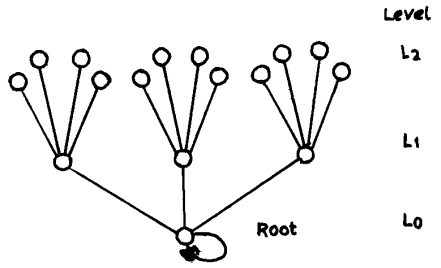


Fig. 1. 4-ary tree structure with 16 nodes.

II. Preliminaries

Definition 1 (m-ary tree structure): We define the m-ary tree structure with m^k nodes (where m and k are integers > 1) as having $k+1$ levels $(L_0, L_1, \cdots, L_k)$ of nodes with the *root node* at L_0 and the *leaf nodes* at L_k. Each node at a level $L_x, x = 1, 2, \cdots, k-1$, connects to m nodes at L_{x+1}, while the root connects to $m-1$ nodes at L_1 and has one connection with itself.

Fig. 1 illustrates a 4-ary tree structure with 16 nodes. If we let $m = 1$ or $k = 1$ in the above definition, a trivial structure results and hence the condition $m > 1$ and $k > 1$. This assumption is used throughout the paper. It can also be observed that with the above definition, a binary or a 2-ary tree structure has 2^k nodes. This differs from the usual graph-theoretic definition which requires an odd number of nodes. However, the proposed structure preserves the properties of the binary tree.

Notation 1: Throughout this paper, it will often be necessary to deal with the binary representation of decimal numbers ranging from 0 to $m^k - 1$. Suppose that m is a power of 2, say $m = 2^\alpha$, where α is any integer > 0. Then each number from 0 to $m^k - 1$ can be represented as an αk-bit (binary) number $x_{\alpha k-1} x_{\alpha k-2} \cdots x_0$ where $x_p \in \{0, 1\}$, for $p = \alpha k - 1$, $\alpha k - 2, \cdots, 0$. We shall express this αk-bit number using the notation $X_{k-1} X_{k-2} \cdots X_0$ where $X_p, p = k-1, k-2, \cdots, 0$, denotes the α-bit string $x_{(p+1)\alpha-1} x_{(p+1)\alpha-2} \cdots x_{p\alpha}$.

Suppose that m is not a power of 2. Let α' be an integer such that $\alpha' = \lceil \log_2 m \rceil$. Let each number from 0 to $m^k - 1$ be represented by $M_{k-1} M_{k-2} \cdots M_0$ in base-m notation, that is, $0 \le M_p \le m-1$, for $p = k-1, k-2, \cdots, 0$. We shall again express each number using the notation $X_{k-1} X_{k-2} \cdots X_0$ where $X_q, q = k-1, k-2, \cdots, 0$, is the α'-bit string $x_{(q+1)\alpha'-1} x_{(q+1)\alpha'-2} \cdots x_{q\alpha'}$, and gives the binary value of M_q.

The difference in the above notation between the two cases (when m is a power of 2 and otherwise) is important. In the former case, $X_p, p = k-1, k-2, \cdots, 0$, is an α-bit string which can assume any one of the 2^α possible values. If m is not a power of 2, X_p is an α'-bit string which can assume any one of m values only, such that its decimal equivalent is between 0 and $m-1$.

Example 1: For $m = 4$ and $k = 3$, each number between 0 and $m^k - 1$ can be represented as the 6-bit ($\alpha k = 6$) number $x_5 x_4 x_3 x_2 x_1 x_0$, or equivalently, $X_2 X_1 X_0$ using Notation 1, where $X_2 = x_5 x_4$, $X_1 = x_3 x_2$, and $X_0 = x_1 x_0$. Each binary string X_2, X_1, or X_0 can assume values 00, 01, 10, or 11.

Let $m = 3$ (hence $\alpha' = 2$) and $k = 3$. Again, using Notation 1, each number from 0 to $m^k - 1$ can be represented as $X_2 X_1 X_0 = x_5 x_4\ x_3 x_2\ x_1 x_0$. However, each binary string X_2, X_1, or X_0, can assume any one of the values 00, 01, or 10 only.

The generalized MIN adopted for the architecture utilizes an interconnection pattern called the m-ary shuffle which is a special case of the pattern defined for delta networks proposed by Patel [49].

Definition 2 (m-ary shuffle connection): Two sets of m^k indexes, each numbered from 0 to $m^k - 1$ and arranged in two columns, respectively, are said to be interconnected by the m-ary shuffle connection if an index i on the left column, $i = 0, 1, \cdots, m^k - 1$, is connected to the index $Shuffl(i)$ on the right column such that[2]

$$Shuffl(i) \equiv (mi + \lfloor i/m^{k-1} \rfloor) \bmod m^k. \tag{2}$$

The interconnection pattern adopted by delta networks is a generalization of the perfect shuffle [50]. The pattern in delta networks is defined as the $a \times b$ shuffle. With the above notation, the m-ary shuffle is equivalent to $m \times m^{k-1}$ shuffle, and the 2-ary shuffle is equivalent to the perfect shuffle. The following result is a direct consequence of (2).

Lemma 1: If $m = 2^\alpha$, then $Shuffl(i)$ is given by the α-bit circular left shift of the αk-bit representation of i.

Fig. 2(a) shows a 4-ary shuffle with 16 indexes. Note that, for instance, index 1 (0001) is connected to 0100, which is the 2-bit circular left shift of 0001. This result is equivalent to that for the $m \times m^{k-1}$ shuffle in [49], where it has been given using base-m notation. It can be easily derived from [49] that if each index is represented as the base-m number $M_{k-1} M_{k-2} \cdots M_0$, then

$$Shuffl(i) = M_{k-2} M_{k-3} \cdots M_0 M_{k-1}. \tag{3}$$

Lemma 2: If m is not a power of 2, then let $\alpha' = \lceil \log_2 m \rceil$. Let each index be represented as $I_{k-1} I_{k-2} \cdots I_0$ according to Notation 1, where $I_q, q = k-1, k-2, \cdots, 0$, is a binary string with decimal value between 0 and $m-1$ (0 and $m-1$ inclusive), then

$$Shuffl(I_{k-1} I_{k-2} \cdots I_0) = I_{k-2} I_{k-3} \cdots I_0 I_{k-1}. \tag{4}$$

The proof of the lemma follows from (3). As an example to illustrate this result, Fig. 2(b) shows the 3-ary shuffle connection with 9 indexes. The indexes are represented according

[2] $\lfloor x \rfloor$ represents the largest integer smaller than or equal to x.

(a) (b)

Fig. 2. Examples of the m-ary shuffle connection: (a) 4-ary shuffle with 16 indexes; (b) 3-ary shuffle with 9 indexes.

to Notation 1. Note that, for instance, 01 10 is connected to $Shuffl(01\ 10) = 10\ 01$.

We now introduce some convenience notations and definitions that we use throughout this paper. $\oplus$ and Λ represent the bitwise EXOR and the bitwise AND operation, respectively. $DEC(X)$ denotes the decimal value of any binary string X. 1_p denotes a string of p binary ones and 0_p denotes a string of p binary zeros.

Definition 3: G_p is the set of all possible p-bit binary numbers. Formally defined,

$$G_p \equiv \{x_{p-1}x_{p-2}\cdots x_0 : x_q \in \{0,1\} \text{ for } q = p-1, p-2, \cdots, 0\}.$$

For example, $G_2 = \{00, 01, 10, 11\}$. It is evident that the cardinality of G_p is 2^p.

Definition 4: Let X be a string of p binary variables $x_{p-1}x_{p-2}\cdots x_0$. Then $G_p(X)$ is the set of all possible combinations of the variables of X in the true and complementary form. Formally defined,

$$G_p(X) \equiv \{x_{p-1}x_{p-2}\cdots x_0 : x_q \in \{x_q, \bar{x}_q\} \text{ for } q = p-1, p-2, \cdots, 0\}$$

where $\bar{x}_q$ represents the binary complement of x_q.

Note that $G_p(X)$ is equivalent to G_p when we assign values to each variable in X.

Definition 5 : Let $X = x_{p-1}x_{p-2}\cdots x_0$ and $Y = y_{p-1}y_{p-2}\cdots y_0$ be two binary strings of p variables each. Consider an arbitrary string $X' = x'_{p-1}x'_{p-2}\cdots x'_0 \in G_p(X)$. Then $\sigma(Y, X')$ is defined as follows:

$$\sigma(Y, X') \equiv y'_{p-1}y'_{p-2}\cdots y'_0$$

$$\text{where, for } q = p-1, p-2, \cdots 0,\ y'_q = \begin{cases} y_q & \text{if } x'_q = x_q \\ \overline{y}_q & \text{if } x'_q = \overline{x}_q \end{cases}.$$

Note that $\sigma(Y, X')$ just denotes a binary string $\in G_p(Y)$ in which each variable has the same form (true or complementary) as that of the corresponding variable in X'.

III. Generalized Architecture for Reconfigurable m-Ary Tree Structures—Case 1: m is a Power of 2

In this section, we present the design and analysis of the architecture for the case when $m = 2^\alpha$, where α is any integer > 0.

A. The Architecture

The system architecture consists of $N = m^k$ nodes, where k is any integer > 1 (Fig. 3). The nodes are interconnected by a MIN of k stages. Each node is an abstraction for a processing element, which consists of a processor with its local memory. Each node is denoted as $P(i)$, $i = 0, 1, \cdots, N-1$, and can be represented as an αk-bit number $i_{\alpha k-1}i_{\alpha k-2}\cdots i_0$, which gives the binary value of the index i. The stages of the MIN are denoted as $S_{k-1}, S_{k-2}, \cdots, S_0$ in order, with S_{k-1} being the leftmost stage. Each stage consists of N/m switching elements (SE's). Each SE has a certain number of ports on its left hand side (LHS) and on its right hand side (RHS). For the sake of simplicity, we shall refer to the LHS ports as inputs and the RHS ports as outputs, although after switching, we assume that the data paths established are bidirectional. Each SE in stages S_{k-1} to S_1 has m inputs, m outputs, and m switching states. The SE's in the last stage are different: each SE in S_0 has m inputs, m^2 outputs, and m switching states.

Consider any stage S_x, $x = k-1, k-2, \cdots, 0$. Since each SE has m inputs and there are $N/m = m^{k-1}$ SE's in S_x, hence the total number of inputs in S_x is $m.m^{k-1} = m^k$. These input lines are numbered $IS_x(0)$, $IS_x(1), \cdots, IS_x(m^k - 1)$ in order, starting at the top. Similarly, the m^k outputs of the SE's in a stage S_x, $x = k-1$, $k-2, \cdots, 1$, are denoted as $OS_x(0), OS_x(1), \cdots, OS_x(m^k - 1)$. Each SE in S_0 has m^2 outputs giving a total of m^{k+1} output lines in S_0. These are numbered $OS_0(0)$, $OS_0(1), \cdots, OS_0(m^{k+1} - 1)$ in order from the top. The set of m^k outputs in S_x, $x = k-1, k-2, \cdots, 1$, is connected to the set of m^k inputs in S_{x-1} by the m-ary shuffle connection.

Each node has $m+1$ terminals, namely, $t_0, t_1, \cdots, t_m$. The terminal t_x of a node $P(i)$, $x = 0, 1, \cdots, m-1$, and $i = 0, 1, \cdots, m^k - 1$, is connected to $OS_0(mi + x)$. Thus a set of m nodes with consecutive indexes is connected to the outputs of each SE in S_0. The terminal t_m of $P(i)$ is connected to $IS_{k-1}(i)$ in stage S_{k-1} in a wrap-around fashion.

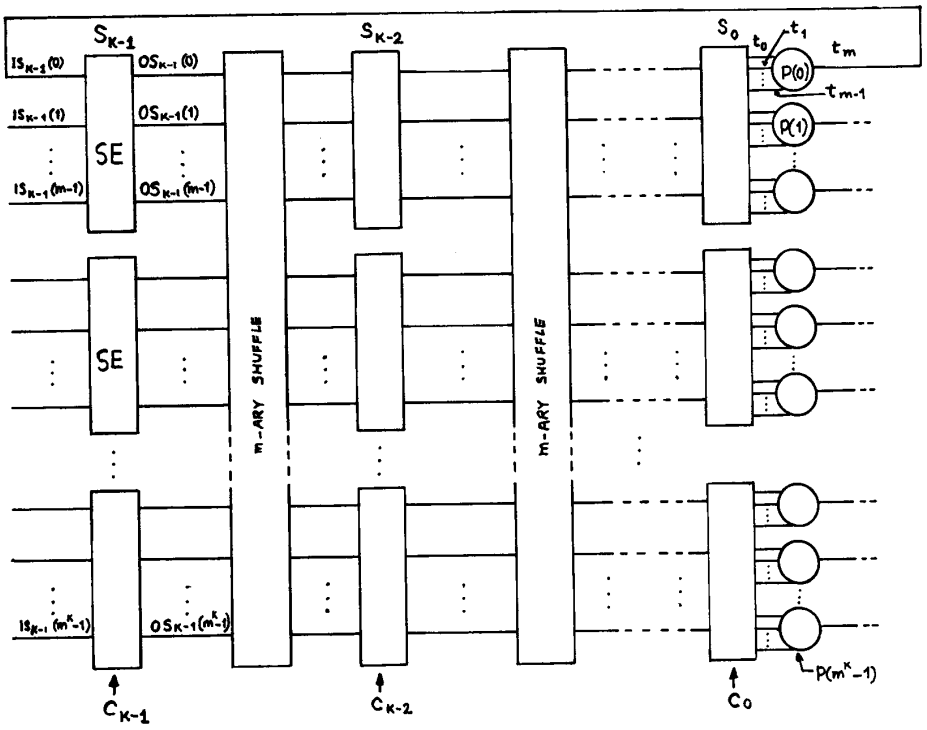


Fig. 3. The generalized architecture for reconfigurable m-ary tree structures ($m = 2^\alpha$).

Each SE is controlled by a set of α control lines. The MIN is *stage-controlled* and synchronous in operation, that is, all the SE's in a particular stage receive the same α-bit code from a single set of α control lines and hence switch to the same state. Thus, the entire network is controlled by an αk-bit *control code* $C = c_{\alpha k-1}c_{\alpha k-2}\cdots c_0$ or equivalently, $C_{k-1}C_{k-2}\cdots C_0$ (using Notation 1), where the α-bit string C_p controls the stage S_p. The control code is issued from a central *Configuration Controller* (CC).

As a particular example of the design, Fig. 4 shows the architecture for reconfigurable 4-ary tree structures with 16 nodes. Note that in the figure (and in many other figures that follow), we denote each node $P(i)$ by its decimal index i only. The eight-node reconfigurable binary tree architecture depicted in Fig. 2 of [1] is another special case of the above design where $m = 2$ and $k = 3$.

B. Switching States of the SE's

We first describe the switching states of the SE's in S_{k-1} to S_1. Each such SE has m inputs and m outputs [Fig. 5(a)]. Let $i_{\alpha-1}i_{\alpha-2}\cdots i_0$ ($j_{\alpha-1}j_{\alpha-2}\cdots j_0$) be the α-bit representation of an arbitrary input (output) belonging to the set of inputs (outputs) $\{0, 1, \cdots, m-1\}$. The switching states of the SE are designed as follows: if $c_{\alpha-1}c_{\alpha-2}\cdots c_0$ is the α-bit code controlling the SE, then each input $i_{\alpha-1}i_{\alpha-2}\cdots i_0$ is connected to the output given by

$$j_{\alpha-1}j_{\alpha-2}\cdots j_0 = (i_{\alpha-1} \oplus c_{\alpha-1})(i_{\alpha-2} \oplus c_{\alpha-2})\cdots(i_0 \oplus c_0). \quad (5)$$

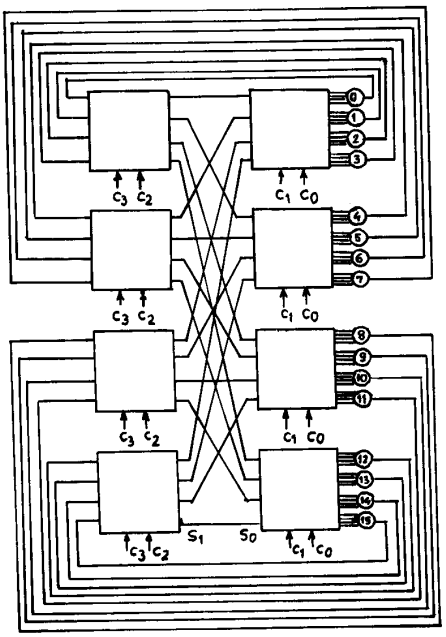
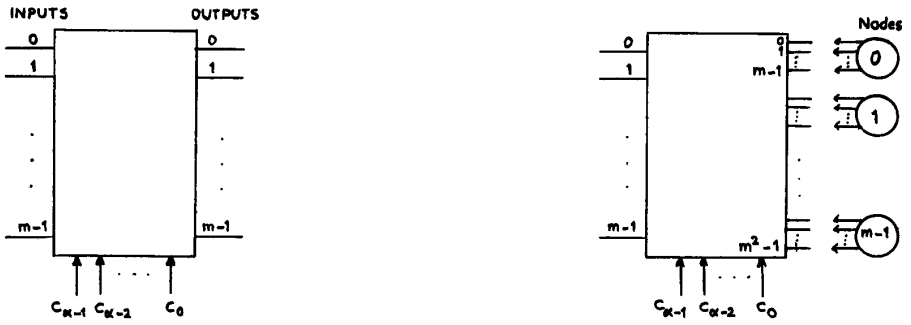


Fig. 4. The architecture for reconfigurable 4-ary tree structures with 16 nodes.

Since the α-bit code can have m possible values, hence the SE has m different switching states, which are given by (5). The switching states of the SE's in S_0 are different. Each



(a) (b)

Fig. 5. General structure of an SE in (a) stages S_{k-1} and (b) stage S_0.

SE has m inputs and m^2 outputs [Fig. 5(b)]. Let the inputs (outputs) be numbered from 0 to $m-1$ (0 to m^2-1) and let c be the decimal value of the α-bit code controlling the SE. Then each switching state is designed such that an input i is connected to the output $(mc+i)$. As mentioned earlier, a set of m nodes with consecutive indexes are connected to the outputs of each SE in S_0. It can be observed in Fig. 5(b) and Fig. 3 that the outputs are partitioned into m sets of m output lines each and the t_0 to t_m terminals of each node are connected to one set of m output lines. It is of interest to determine the nodes which get connected in each state rather than the individual output lines. Let $j_{\alpha-1}j_{\alpha-2}\cdots j_0$ represent an arbitrary node of the set of m nodes which are connected to the outputs of the SE. Then, for each value of the code $c_{\alpha-1}c_{\alpha-2}\cdots c_0$ controlling the SE, all the inputs are connected to the node given by

$$j_{\alpha-1}j_{\alpha-2}\cdots j_0 = c_{\alpha-1}c_{\alpha-2}\cdots c_0. \tag{6}$$

The switching states of the SE's in the reconfigurable binary tree architecture [1] can now be obtained as a particular case of the above design with $m=2$. Fig. 6 shows the switching states of the SE's. As another example, Fig. 7 depicts the switching states of the SE's in the architecture for reconfigurable 4-ary tree structures. In Fig. 7(b), which shows the switching states of an SE in S_0, we have denoted the four nodes that are connected to the outputs as 00, 01, 10, and 11. According to (6), for the code 00 controlling the SE, all the inputs are connected to the node 00; for control code 01, all inputs are connected to the node 01; and so on.

We now derive the equation for switching in a stage by considering all the inputs and outputs in that stage. Denote an arbitrary input $IS_p(i)$ of the set of m^k inputs of a stage $S_p, p = k-1, k-2, \cdots, 1$, by the αk-bit address $i^{(p)}_{\alpha k-1}i^{(p)}_{\alpha k-2}\cdots i^{(p)}_0$ which gives the binary value of i. Similarly, denote the arbitrary output $OS_p(i)$ of S_p by $j^{(p)}_{\alpha k-1}j^{(p)}_{\alpha k-2}\cdots j^{(p)}_0$. Let the α-bit code controlling S_p be $c^{(p)}_{\alpha-1}c^{(p)}_{\alpha-2}\cdots c^{(p)}_0$. Since the MIN is controlled by the αk-bit control code $C = c_{\alpha k-1}c_{\alpha k-2}\cdots c_0$, with each α-bit string of C controlling a stage, it is easy to verify that

$$c^{(p)}_{\alpha-1}c^{(p)}_{\alpha-2}\cdots c^{(p)}_0 = c_{(p+1)\alpha-1}c_{(p+1)\alpha-2}\cdots c_{p\alpha}. \tag{7}$$

Using (5) and (7), the equation for switching in S_p can be derived as follows:

$$\begin{aligned} &j^{(p)}_{\alpha k-1}j^{(p)}_{\alpha k-2}\cdots j^{(p)}_0 \\ &\quad = i^{(p)}_{\alpha k-1}i^{(p)}_{\alpha k-2}\cdots i^{(p)}_{\alpha}(i^{(p)}_{\alpha-1}\oplus c^{(p)}_{\alpha-1})(i^{(p)}_{\alpha-2}\oplus c^{(p)}_{\alpha-2}) \\ &\qquad \cdots (i^{(p)}_0 \oplus c^{(p)}_0) \\ &\quad = i^{(p)}_{\alpha k-1}i^{(p)}_{\alpha k-2}\cdots i^{(p)}_{\alpha}(i^{(p)}_{\alpha-1}i^{(p)}_{\alpha-2}\cdots i^{(p)}_0 \\ &\qquad \oplus c_{(p+1)\alpha-1}c_{(p+1)\alpha-2}\cdots c_{p\alpha}). \end{aligned} \tag{8}$$

Similarly, if we denote an arbitrary input of S_0 as $i^{(0)}_{\alpha k-1}i^{(0)}_{\alpha k-2}\cdots i^{(0)}_0$ and the code controlling S_0 as $c^{(0)}_{\alpha-1}c^{(0)}_{\alpha-2}\cdots c^{(0)}_0$, then, from (6) and (7),

$$j_{\alpha k-1}j_{\alpha k-2}\cdots j_0 = i^{(0)}_{\alpha k-1}i^{(0)}_{\alpha k-2}\cdots i^{(0)}_{\alpha}c_{\alpha-1}c_{\alpha-2}\cdots c_0 \tag{9}$$

where the LHS denotes the nodes to which the inputs of S_0 are connected. Using Notation 1, we can rewrite (8) and (9), respectively, as follows:

$$J^{(p)}_{k-1}J^{(p)}_{k-2}\cdots J^{(p)}_1 J^{(p)}_0 = I^{(p)}_{k-1}I^{(p)}_{k-2}\cdots I^{(p)}_1(I^{(p)}_0 \oplus C_p) \tag{10}$$

$$J_{k-1}J_{k-2}\cdots J_1 J_0 = I^{(0)}_{k-1}I^{(0)}_{k-2}\cdots I^{(0)}_1 C_0 \tag{11}$$

where $X^{(p)}_q$ denotes the α-bit string $x^{(p)}_{(q+1)\alpha-1}x^{(p)}_{(q+1)\alpha-2}\cdots x^{(p)}_{q\alpha}$.

C. Generalized Reconfiguration Equation

The architecture described above is able to assume distinct m-ary tree configurations among the nodes. The CC is responsible for establishing the configurations. For each αk-bit control code issued by the CC, a distinct m-ary TC is obtained. This interesting property follows from the theorem stated below.

Theorem 1: When an αk-bit control code C is issued to the MIN, each node $P(i)$ establishes connection with a node $P(j)$ which is given by the following *generalized reconfiguration equation* (GRE):

$$P(j) = (CRS^{\alpha}[P(i)]\Lambda B) \oplus C \tag{12}$$

Fig. 6. Switching states of the SE's in the reconfigurable binary tree architecture: (a) SE in S_{k-1} to S_1 and (b) SE in S_0.

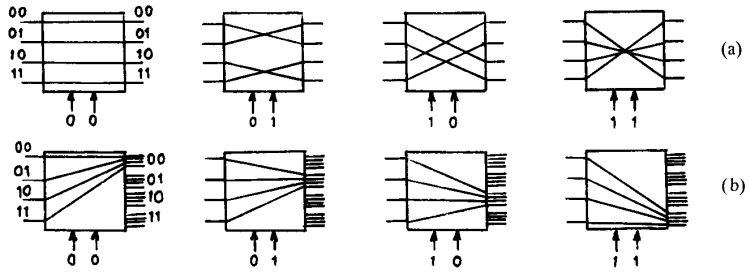


Fig. 7. Switching states of the SE's in the architecture with $m = 4$: (a) SE in S_{k-1} to S_1 and (b) SE in S_0.

where $CRS^{\alpha}[P(i)]$ is the α-bit circular right shift of $P(i)$, and B is the αk-bit number

$$= B_{k-1}B_{k-2}\cdots B_1B_0 = 1_{\alpha}1_{\alpha}\cdots 1_{\alpha}0_{\alpha}.$$

Proof: Start from the t_m terminal of an arbitrary node $P(i) = I_{k-1}I_{k-2}\cdots I_0$ and trace the path through the MIN to the node $P(j)$ to which it is connected. The switching at S_{k-1} is controlled by the α-bit string C_{k-1} of C. Hence, from (10), the path address at the output of S_{k-1} is given by

$$J_{k-1}^{(k-1)}J_{k-2}^{(k-1)}\cdots J_1^{(k-1)}J_0^{(k-1)} = I_{k-1}I_{k-2}\cdots I_1(I_0 \oplus C_{k-1}). \tag{13}$$

From Lemma 1 and (13), the path address at the input of S_{k-2} is

$$I_{k-1}^{(k-2)}I_{k-2}^{(k-2)}\cdots I_0^{(k-2)} = I_{k-2}I_{k-3}\cdots I_1(I_0 \oplus C_{k-1})I_{k-1}. \tag{14}$$

Proceeding in a similar manner stage by stage, the path address at the input of S_0 is

$$I_{k-1}^{(0)}I_{k-2}^{(0)}\cdots I_0^{(0)} = (I_0 \oplus C_{k-1})(I_{k-1} \oplus C_{k-2}) \cdot (I_{k-2} \oplus C_{k-3})\ldots(I_2 \oplus C_1)I_1. \tag{15}$$

The switching at S_0 takes place according to (11) and the destination node $P(j)$ is given by

$$\begin{aligned} J_{k-1}J_{k-2}\cdots J_1J_0 &= I_{k-1}^{(0)}I_{k-2}^{(0)}\cdots I_1^{(0)}C_0 \\ &= (I_0 \oplus C_{k-1})(I_{k-1} \oplus C_{k-2}) \\ &\quad \cdot (I_{k-2} \oplus C_{k-3})\cdots(I_2 \oplus C_1)C_0 \text{ [from (15)]} \\ &= (I_0I_{k-1}I_{k-2}\cdots I_2I_1 \wedge 1_{\alpha}1_{\alpha}1_{\alpha} \\ &\quad \cdots 1_{\alpha}0_{\alpha}) \oplus C_{k-1}C_{k-2}\cdots C_1C_0 \\ &= (CRS^{\alpha}[I_{k-1}I_{k-2}\cdots I_0] \wedge B_{k-1}B_{k-2} \\ &\quad \cdots B_1B_0) \oplus C_{k-1}C_{k-2}\cdots C_1C_0. \end{aligned} \tag{16}$$

■

Example 2: In the architecture with $m = 4$ and $k = 2$ (Fig. 4), let C = 01 00. Then the node $P(3)$, for instance, is connected through the MIN to the node given by $(CRS^2[00\ 11]\Lambda 11\ 00) \oplus 01\ 00 = 10\ 00$ (that is, node $P(8)$).

Equation (12) is a generalization of the reconfiguration equation (1) which was derived for the binary tree architecture [1]. It is significant in that it forms the basis for the proof that the architecture is able to assume distinct m-ary TC's.

D. Proof of Reconfiguration

In this section, we prove the reconfigurability of the architecture.

Definition 6: In the GRE (12), $P(j)$ is called the *successor* of $P(i)$. $P(i)$ is the *predecessor* of $P(j)$. The set of all predecessors of $P(j)$ is denoted $PRED(P(j))$.

Lemma 3: For any control code $C = C_{k-1}C_{k-2}\cdots C_0$, the predecessors of a node $P(j) = J_{k-1}J_{k-2}\cdots J_0$ are given by

$$PRED(P(j)) = \begin{cases} \{D_{k-1}D_{k-2}\cdots D_2D_1^*D_0 : D_1^* \in \boldsymbol{G_\alpha}\} & \text{if } J_0 = C_0 \\ \varnothing & \text{otherwise.} \end{cases}$$

$$\text{where } D_q = \begin{cases} J_{q-1} \oplus C_{q-1} & \text{for } q = k-1, k-2, \cdots, 2 \\ J_{k-1} \oplus C_{k-1} & \text{for } q = 0 \end{cases}. \tag{17}$$

The proof of the above lemma can be obtained by solving (12) for $P(i)$. Note that since D_1^* is an α-bit string which can take any one of 2^α possible values, hence any node which satisfies the condition $J_0 = C_0$ has $2^\alpha = m$ predecessors, each of which differ only in the string D_1^*. Note also that if $k = 2$ in (17), only the relation $D_0 = J_1 \oplus C_1$ is valid and (17) reduces to

$$PRED(J_1J_0) = \begin{cases} \{D_1^*D_0 : D_1^* \in \boldsymbol{G_\alpha}\} & \text{if } J_0 = C_0 \\ \varnothing & \text{otherwise} \end{cases}. \tag{18}$$

Similar changes arising due to $k = 2$ must be kept in mind while dealing with other equations too.

Example 3 : Let $m = 4$ and $k = 3$. Then the predecessors of a node $P(j) = j_5j_4\ j_3j_2\ j_1j_0 = J_2J_1J_0$ are given by

$$PRED(J_2J_1J_0) = \begin{cases} \{(J_1 \oplus C_1)D_1^*(J_2 \oplus C_2) : D_1^* \in \boldsymbol{G_2}\} & \text{if } J_0 = C_0 \\ \varnothing & \text{otherwise} \end{cases}.$$

For instance, if the control code $C_2C_1C_0 =$ 01 11 10, then the node 00 01 10 has four predecessors given by $\{(01 \oplus 11)\ D_1^*\ (00 \oplus 01) : D_1^* \in \boldsymbol{G_2}\}$ = {10 00 01, 10 01 01, 10 10 01, 10 11 01}.

Lemma 4: For any control code $C = c_{\alpha k-1}c_{\alpha k-2}\cdots c_0 = C_{k-1}C_{k-2}\cdots C_0$, there exists a single node $R = r_{\alpha k-1}r_{\alpha k-2}\cdots r_0 = R_{k-1}R_{k-2}\cdots R_0$ which has a connection with itself and is given by

$$R_p = \begin{cases} C_p & p = 0 \\ C_0 \oplus C_{k-1} \oplus C_{k-2} \oplus \cdots \oplus C_p & p = k-1, k-2, \cdots 1 \end{cases}. \tag{19}$$

Proof: A node R that has a connection with itself satisfies the condition $P(i) = P(j) = R$ in the GRE. Thus, R is determined by equating the bits on both sides of the equation $R = (CRS^\alpha[R]\Lambda B) \oplus C$. ■

For example, with $m = 4$, $k = 3$, and $C =$ 00 01 10, $R = R_2R_1R_0 = (10 \oplus 00)(10 \oplus 00 \oplus 01)(10) =$ 10 11 10. It will be evident shortly that R forms the root of the TC. Furthermore, it can be easily verified from (19) using switching theoretic principles that the following results (Lemmas 5–7) are valid. Lemma 8 can be derived from Lemmas 5–7.

Lemma 5:

$$R_p \oplus C_p = \begin{cases} R_{p+1} & p = k-2, k-3, \cdots 1 \\ R_0 & p = k-1 \end{cases}. \tag{20}$$

Lemma 6:

$$r_{p\alpha-q} \oplus c_{p\alpha-q} = \begin{cases} r_{(p+1)\alpha-q} & p = k-1, k-2, \cdots, 2, q = 1, 2, \cdots, \alpha \\ r_{\alpha-q} & p = k, q = 1, 2, \cdots, \alpha \end{cases}. \tag{21}$$

Lemma 7:

$$\overline{r}_{p\alpha-q} \oplus c_{p\alpha-q} = \begin{cases} \overline{r}_{(p+1)\alpha-q} & p = k-1, k-2, \cdots, 2, q = 1, 2, \cdots, \alpha \\ \overline{r}_{\alpha-q} & p = k, q = 1, 2, \cdots, \alpha \end{cases}. \tag{22}$$

Lemma 8: Let R'_p be a string of α binary variables such that $R'_p \in \boldsymbol{G_\alpha}(R_p)$, for some $p = k-1, k-2, \cdots, 1$. Then

$$R'_p \oplus C_p = \begin{cases} \sigma(R_{p+1}, R'_p) & p = k-2, k-3, \cdots 1 \\ \sigma(R_0, R'_p) & p = k-1 \end{cases}. \tag{23}$$

Example 4: Consider the node R given by $R_2R_1R_0 = r_5r_4\ r_3r_2\ r_1r_0$. Let $R'_1 = r_3\overline{r}_2$. Then, from Lemma 8, the operation $R'_1 \oplus C_1$ gives $\sigma(R_2, R'_1) = r_5\overline{r}_4$.

Theorem 2: Each value of the control code C establishes a distinct $(k+1)$-level m-ary tree configuration among the nodes.

Proof: Part 1: Each value of C establishes a $(k+1)$-level m-ary TC.

For each value of C, there is a single node R which has a connection with itself (Lemma 4). Let R be at level L_0. Since the condition $R_0 = C_0$ is true (Lemma 4), hence the node has m predecessors which are given by Lemma 3. But, by defintion, R is always one of its predecessors. Hence, the set of distinct predecessors of R (that is, the set not containing R) is

$$\begin{aligned} &\{(R_{k-2} \oplus C_{k-2})(R_{k-3} \oplus C_{k-3})\cdots \\ &\quad (R_1 \oplus C_1)R_1^{**}(R_{k-1} \oplus C_{k-1}) : R_1^{**} \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})\backslash\{R_1\}\} \\ &= \{R_{k-1}R_{k-2}\cdots R_2R_1^{**}R_0 : R_1^{**} \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})\backslash\{R_1\}\} \\ &\qquad \text{(from Lemma 5)} \end{aligned}$$

where $\backslash$ is the set difference operator. These $m-1$ nodes are the nodes at level L_1. Consider an arbitrary member $R_{k-1}R_{k-2}\cdots R_2R''_1R_0$ of this set, where $R''_1 \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})\backslash\{R_1\}$. Observe that this node differs from R only in the string R''_1, which has one or more variables of R_1 in the complementary form. Now, using Lemmas 3 and 5,

$$\begin{aligned} &PRED(R_{k-1}R_{k-2}\cdots R_2R''_1R_0) \\ &\quad = \{(R_{k-2} \oplus C_{k-2})(R_{k-3} \oplus C_{k-3})\cdots \\ &\qquad (R_2 \oplus C_2)(R''_1 \oplus C_1)R_1^*(R_{k-1} \oplus C_{k-1}) \\ &\qquad : R_1^* \in \boldsymbol{G_\alpha}(R_1)\} \\ &\quad = \{R_{k-1}R_{k-2}\cdots R_3(R''_1 \oplus C_1)R_1^*R_0 \\ &\qquad : R_1^* \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})\}. \end{aligned}$$

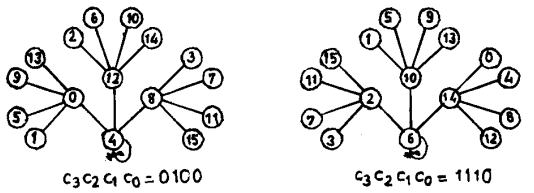


Fig. 8. Two of the possible sixteen 4-ary TC's realized by the architecture of Fig. 4 with (a) C = 01 00 and (b) C = 11 10.

Here $R_1'' \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})\backslash\{R_1\} \subset \boldsymbol{G_\alpha}(\boldsymbol{R_1})$. Hence from Lemma 8, $R_1'' \oplus C_1 = \sigma(R_2, R_1'')$. Let $\sigma(R_2, R_1'') = R_2''$ which $\in \boldsymbol{G_\alpha}(\boldsymbol{R_2})\backslash\{R_2\}$. Making these substitutions.

$$PRED(R_{k-1}R_{k-2}\cdots R_2R_1''R_0) = \{R_{k-1}R_{k-2}\cdots R_3R_2''R_1^*R_0 : R_1^* \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})\}.$$

In other words, each predecessor has a string R_2'' which has its variables in the same pattern (true or complemented) as those of R_1''. Thus, every node $R_{k-1}R_{k-2}\cdots R_2R_1''R_0$ at L_1 has m predecessors, each of which can be expressed in the form $R_{k-1}R_{k-2}\cdots R_3R_2''R_1'R_0$ where $R_2'' \in \boldsymbol{G_\alpha}(\boldsymbol{R_2})\backslash\{R_2\}$ and $R_1' \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})$. Proceeding in a similar manner, the nodes at various levels can be expressed as follows:

Level	Node
L_0	$R_{k-1}R_{k-2}R_{k-3}\cdots R_4R_3R_2R_1R_0$
L_1	$R_{k-1}R_{k-2}R_{k-3}\cdots R_4R_3R_2R_1''R_0$
L_2	$R_{k-1}R_{k-2}R_{k-3}\cdots R_4R_3R_2''R_1'R_0$
L_3	$R_{k-1}R_{k-2}R_{k-3}\cdots R_4R_3''R_2'R_1'R_0$
.	.
L_{k-1}	$R_{k-1}''R_{k-2}'R_{k-3}'\cdots R_4'R_3'R_2'R_1'R_0$
L_k	$R_{k-1}'R_{k-2}'R_{k-3}'\cdots R_4'R_3'R_2'R_1'R_0''$

where $R_x'' \in \boldsymbol{G_\alpha}(\boldsymbol{R_x})\backslash\{R_x\}$ and $R_x' \in \boldsymbol{G_\alpha}(\boldsymbol{R_x})$ for $x = 0, 1, \cdots, k-1$.

It is evident from the above that the node at L_0 is the root node which has $m-1$ distinct predecessors. Each node at L_x, $x = 1, 2, \cdots, k-1$, has m predecessors at L_{x+1}. Each node at L_k does not satisfy the condition $J_0 = C_0$ of Lemma 3 and hence it is a leaf node. Furthermore, any node at L_p is distinct from a node at $L_q, p \neq q$. This defines an m-ary TC of $k+1$ levels.

Part 2: The TC is distinct for a distinct value of C. This follows from the fact that R is distinct for a distinct value of C (by Lemma 4) ■.

Thus, the architecture can assume $2^{\alpha k} = m^k$ distinct m-ary tree configurations, one for each value of the αk-bit code C. Fig. 8 shows two of the 16 possible 4-ary TC's obtained with the architecture of Fig. 4. Fig. 9 shows the eight binary TC's that are generated in an architecture with $m = 2$ and $k = 3$.

E. Synthesis of the m-ary Tree Configuration from the Root Node

The analysis of the GRE that was carried out in the previous section in order to prove the theorems and lemmas indicate that the entire m-ary TC can be synthesized from the root node, for any given value of the control code C. C determines the root node and any other node in the configuration can be expressed as a function of the root.

As an example, consider the architecture for $m = 4$ and $k = 2$. For a given value of $C = C_1C_0 = c_3c_2c_1c_0$, the root $R = R_1R_0 = r_3r_2r_1r_0 = (c_3 \oplus c_1)(c_2 \oplus c_0)c_1c_0$. The set of distinct predecessors of the root is $\{R_1^{**}R_0 : R_1^{**} \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})\backslash\{R_1\}\}$ (proof of Theorem 2). Thus, the nodes at L_1 are $\bar{r}_3r_2r_1r_0$, $r_3\bar{r}_2r_1r_0$, and $\bar{r}_3\bar{r}_2r_1r_0$. Consider any of these nodes, for instance, $\bar{r}_3r_2r_1r_0$. Its predecessors are given by $\{R_1^*\sigma(R_0, R_1'') : R_1^* \in \boldsymbol{G_\alpha}(\boldsymbol{R_1})\}$ where $R_1'' = \bar{r}_3r_2$. Hence, $\sigma(R_0, R_1'') = \bar{r}_1r_0$ and the predecessors are the nodes $r_3r_2\bar{r}_1r_0$, $\bar{r}_3r_2\bar{r}_1r_0$, $r_3\bar{r}_2\bar{r}_1r_0$, and $\bar{r}_3\bar{r}_2\bar{r}_1r_0$. These nodes are at level L_2. The predecessors of $r_3\bar{r}_2r_1r_0$ and $\bar{r}_3\bar{r}_2r_1r_0$ can be found in a similar manner. Fig. 10 shows the 4-ary TC for the above example. It can be verified that the TC's in Figs. 8(a) and (b) conform to this synthesis procedure by substituting the corresponding values of R.

IV. Generalized Architecture for Reconfigurable m-Ary Tree Structures—Case 2: m is not a Power of 2

We now present the design and analysis of the architecture when m is not a power of 2. Let α' be the integer $\lceil \log_2 m \rceil$.

The system architecture is very similar to that of Case 1. The $N(= m^k)$ nodes are interconnected by a MIN of k stages $(S_{k-1}, S_{k-2}, \cdots, S_0)$. Each stage consists of N/m SE's and the different stages are interconnected by the m-ary shuffle connection. Throughout this section, we shall represent the nodes, and the inputs and outputs of each stage of the MIN using Notation 1. For example, if $m = 3$ and $k = 2$, the 9 nodes are represented as 00 00, 00 01, 00 10, 01 00, 01 01, 01 10, 10 00, 10 01, and 10 10. Note that the corresponding decimal values for the node representations are 0, 1, 2, 4, 5, 6, 8, 9, and 10.

The difference with Case 1 is in the switching states of the SE's. Each SE in S_{k-1} to S_1 has m inputs, m outputs, and is controlled by an α'-bit code. If each input (output) is represented in binary as $i_{\alpha'-1}i_{\alpha'-2}\cdots i_0$ $(j_{\alpha'-1}j_{\alpha'-2}\cdots j_0)$ and the α'-bit code controlling the SE is $c_{\alpha'-1}c_{\alpha'-2}\cdots c_0$, then the equation for switching is as follows:

$$j_{\alpha'-1}j_{\alpha'-2}\cdots j_0 = \begin{cases} (i_{\alpha'-1}i_{\alpha'-2}\cdots i_0) \oplus (c_{\alpha'-1}c_{\alpha'-2}\cdots c_0) \\ \qquad \text{if } DEC((i_{\alpha'-1}i_{\alpha'-2}\cdots i_0) \\ \qquad \oplus (c_{\alpha'-1}c_{\alpha'-2}\cdots c_0)) < m. \\ i_{\alpha'-1}i_{\alpha'-2}\cdots i_0 \qquad \text{otherwise} \end{cases} \tag{24}$$

The SE has $2^{\alpha'}$ switching states, one for each value of $c_{\alpha'-1}c_{\alpha'-2}\cdots c_0$.

Each SE in S_0 has m inputs, m^2 outputs, and is controlled by an α'-bit code which can assume only values 0 to $m-1$ in decimal. As in Case 1, a set of m nodes with consecutive indexes connects to the outputs of the SE and the switching is similar to that in Case 1. That is, for each value of the code $c_{\alpha'-1}c_{\alpha'-2}\cdots c_0$ controlling the SE, all the inputs are

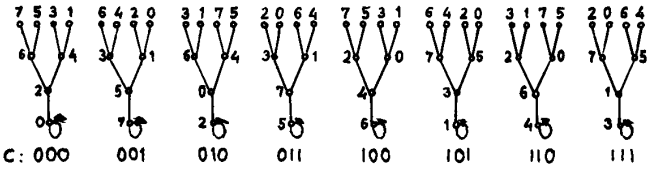


Fig. 9. The eight distinct binary tree configurations realized by the architecture with $m = 2$ and $k = 3$.

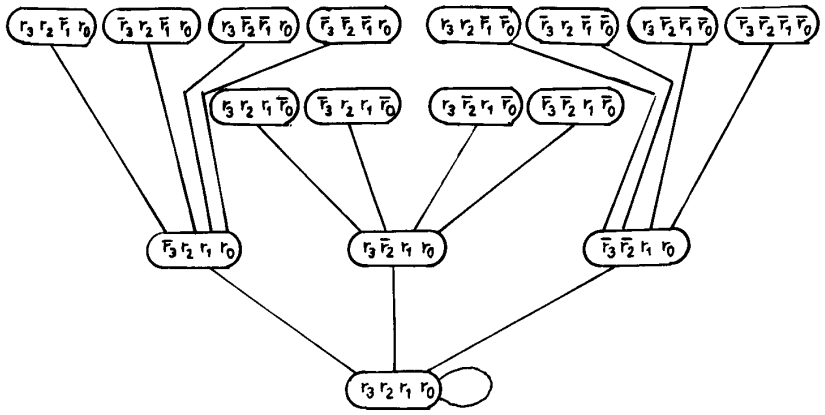


Fig. 10. Synthesis of the tree configuration from the root node.

(a) (b)

Fig. 11. Switching states of the SE's in the architecture with $m = 3$: (a) SE in S_{k-1} to S_1 and (b) SE in S_0.

connected to the node given by

$$j_{\alpha'-1}j_{\alpha'-2}\cdots j_0 = c_{\alpha'-1}c_{\alpha'-2}\cdots c_0. \tag{25}$$

Note that since the control code can have only m values, the number of switching states of the SE is m and not $2^{\alpha'}$.

Example 5: Fig. 11 shows the switching states of the SE's for $m = 3$. Consider, for instance, the switching state of an SE in S_{k-1} to S_1 for the control code 01. According to (24), the input 00 is connected to the output 00 ⊕ 01 = 01; the input 01 is connected to 01 ⊕ 01 = 00; while the input 10 is connected to the output 10 because DEC(10 ⊕ 01) ≮ 3.

All the SE's in a particular stage receive the same α'-bit code and hence have the same state. Thus the entire MIN is controlled by an $\alpha' k$-bit control code $C = c_{\alpha' k-1}c_{\alpha' k-2}\cdots c_0 = C_{k-1}C_{k-2}\cdots C_0$. Note that C_{k-1}, $C_{k-2},\cdots,C_1$ can assume one of $2^{\alpha'}$ values while C_0 can have one of m values only. Thus C can have $2^{\alpha'(k-1)}.m$ different values.

Fig. 12 illustrates the architecture for reconfigurable 3-ary tree structures with 9 nodes. The nodes are numbered in decimal with values equal to the corresponding binary notation.

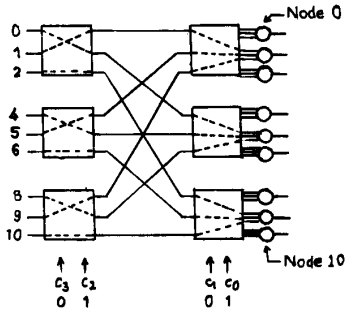


Fig. 12. The architecture for reconfigurable 3-ary tree structures with 9 nodes. The switching states are shown for the control code 01 01.

If we denote an arbitrary input (output) of the set of m^k inputs (outputs) of a stage S_p, $p = k-1, k-2,\cdots,1$, as $I_{k-1}^{(p)}I_{k-2}^{(p)}\cdots I_0^{(p)}$ ($J_{k-1}^{(p)}J_{k-2}^{(p)}\cdots J_0^{(p)}$), we can derive the following equation for switching in a stage in a manner similar to (10) in Case 1:

$$J_{k-1}^{(p)}J_{k-2}^{(p)}\cdots J_1^{(p)}J_0^{(p)} = I_{k-1}^{(p)}I_{k-2}^{(p)}\cdots I_1^{(p)}E_0^{(p)}$$

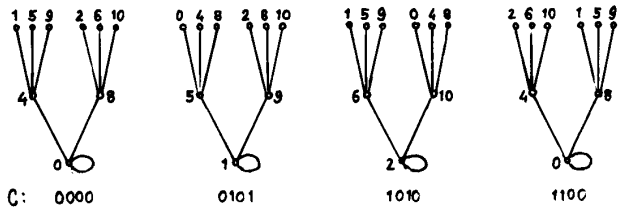


Fig. 13. Four of the possible twelve 3-ary TC's realized by the architecture of Fig. 12.

$$\text{where } E_0^{(p)} = \begin{cases} I_0^{(p)} \oplus C_p & \text{if } DEC(I_0^{(p)} \oplus C_p) < m \\ I_0^{(p)} & \text{otherwise} \end{cases}. \tag{26}$$

Similarly, if $I_{k-1}^{(0)} I_{k-2}^{(0)} \cdots I_0^{(0)}$ is the representation of an input of S_0, then

$$J_{k-1} J_{k-2} \cdots J_0 = I_{k-1}^{(0)} I_{k-2}^{(0)} \cdots I_1^{(0)} C_0 \tag{27}$$

where the LHS denotes the nodes to which the inputs of S_0 are connected.

Note that (26) is identical to (10) if $DEC(I_0^{(p)} \oplus C_p) < m$ and (27) is identical to (11) except that C_0 can have only m (and not $2^{\alpha'}$) values. Furthermore, it can be verified from (26) and using the input-output symmetry property of the SE's that the following equation is valid:

$$I_{k-1}^{(p)} I_{k-2}^{(p)} \cdots I_1^{(p)} I_0^{(p)} = J_{k-1}^{(p)} J_{k-2}^{(p)} \cdots J_1^{(p)} F_0^{(p)}$$
$$\text{where } F_0^{(p)} = \begin{cases} J_0^{(p)} \oplus C_p & \text{if } DEC(J_0^{(p)} \oplus C_p) < m \\ J_0^{(p)} & \text{otherwise} \end{cases}. \tag{28}$$

Notation 2: The equation

$$Z = \begin{cases} X \oplus Y & \text{if } DEC(X \oplus Y) < m \\ X & \text{otherwise} \end{cases}$$

where X, Y, and Z are three α'-bit strings, can be expressed as follows:

$$Z = X \oplus \hat{Y} \text{ where } \hat{Y} = \begin{cases} Y & \text{if } DEC(X \oplus Y) < m \\ 0_{\boldsymbol{\alpha'}} & \text{otherwise} \end{cases}.$$

Using this notation, $E_0^{(p)}$ in (26) can be expressed as $I_0^{(p)} \oplus \hat{C}_p$ where $\hat{C}_p = C_p$ or $0_{\alpha'}$, depending upon whether $DEC(I_0^{(p)} \oplus C_p) < m$ or otherwise. Similarly, $F_0^{(p)}$ in (28) can be written as $J_0^{(p)} \oplus \hat{C}_p$ where $\hat{C}_p = C_p$ (if $DEC(J_0^{(p)} \oplus C_p) < m$) or $0_{\alpha'}$ (otherwise).

The reconfiguration equation for the architecture can now be derived.

Theorem 3: Each node $P(i)$ establishes connection with a node $P(j)$ given by

$$P(j) = (CRS^{\alpha'}[P(i)] \Lambda B) \oplus C_{HAT} \tag{29}$$

where $CRS^{\alpha'}$ is the α'-bit circular right shift of $P(i)$, $B = B_{k-1} B_{k-2} \cdots B_1 B_0 = 1_{\alpha'} 1_{\alpha'} \cdots 1_{\alpha'} 0_{\alpha'}$, and $C_{HAT} = \hat{C}_{k-1} \hat{C}_{k-2} \cdots \hat{C}_1 C_0$.

Proof: Can be obtained in a manner similar to that of Theorem 1, using (26), (27), Notation 2, and Lemma 2.

Example 6: In Fig. 13, $C = C_1 C_0 = 01\ 01$. The node $P(6)$, for instance, is connected to $P(j) = (CRS^2[0110] \Lambda 1100) \oplus \hat{C}_1 C_0 = 1000 \oplus \hat{C}_1 C_0$. Now $\hat{C}_1 = 00$ since $10 \oplus C_1 \not< 3$. Hence $P(j) = 1000 \oplus 0001 = 1001$ (node $P(9)$).

Results with regard to the predecessors of a node $P(j) = J_{k-1} J_{k-2} \cdots J_0$ and the node $R = R_{k-1} R_{k-2} \cdots R_0$ which has a connection with itself can also be derived by following a procedure similar to Case 1 and using (26)–(29).

Lemma 9:

$$PRED(P(j)) = \begin{cases} \{D_{k-1} D_{k-2} \cdots D_2 D_1^* D_0 : D_1^* \in \boldsymbol{G_{\alpha'}} \\ \quad \text{and } DEC(D_1^*) < m\} & \text{if } J_0 = C_0 \\ \varnothing & \text{otherwise.} \end{cases}$$
$$\text{where } D_q = \begin{cases} J_{q-1} \oplus \hat{C}_{q-1} & q = k-1, k-2, \cdots, 2 \\ J_{k-1} \oplus \hat{C}_{k-1} & q = 0 \end{cases}.$$

Thus, any node which satisfies the condition $J_0 = C_0$ has m predecessors.

Lemma 10:

$$R_p = \begin{cases} C_p & p = 0 \\ R_0 \oplus \hat{C}_p & p = k-1 \\ R_{p+1} \oplus \hat{C}_p & p = k-2, k-3, \cdots 1 \end{cases}.$$

The following result can be proved from Lemma 9 and making use of the fact that one of the predecessors of R is R itself.

Lemma 11:

$$R_p \oplus \hat{C}_p = \begin{cases} R_{p+1} & p = k-2, k-3, \cdots 1 \\ R_0 & p = k-1 \end{cases}.$$

It can be observed in the preceding results that they are similar to those obtained for Case 1, with the exception of C_p being replaced by $\hat{C}_p$ at the appropriate places.

Lemma 12: Let $SUCC(P(i))_C$ represent the successor of a node $P(i)$ for the control code C. If $C1$ and $C2$ are any two distinct control codes, then there exists at least one node $P(x)$ such that $SUCC(P(x))_{C1} \neq SUCC(P(x))_{C2}$.

Proof: From (29),

$$SUCC(P(i)) = (I_0 \oplus \hat{C}_{k-1})(I_{k-1} \oplus \hat{C}_{k-2}) \cdot (I_{k-2} \oplus \hat{C}_{k-3}) \cdots (I_2 \oplus \hat{C}_1) C_0. \tag{30}$$

Let $C1$ and $C2$ differ in the binary strings $C1_p$ and $C2_p$ only. That is, let $C1 = C_{k-1} C_{k-2} \cdots C_{p+1} C1_p C_{p-1} \cdots C_0$ and $C2 = C_{k-1} C_{k-2} \cdots C_{p+1} C2_p C_{p-1} \cdots C_0$ for any $p = k-1, k-2, \cdots, 0$.

$p = 0$: From (30), the successors of any node for $C1$ and $C2$ are distinct and hence the proof.

$p \neq 0$: Let $C1_p \neq 0_{\alpha'}$. There exists at least one node $P(x)$ for which $DEC(X_{p+1} \oplus \hat{C}1_p) < m$. Hence its successor for $C1$ contains the string $X_{p+1} \oplus C1_p$, while its successor for $C2$ contains the string $X_{p+1} \oplus \hat{C}2_p$. Whatever be the value of $\hat{C}2_p(0_{\alpha'}$ or otherwise), $X_{p+1} \oplus \hat{C}2_p \neq X_{p+1} \oplus C1_p$, and hence the inequality is valid. The same logic can be used to prove the lemma when $\hat{C}1_p = 0_{\alpha'}$.

Theorem 4 : Each value of C establishes a distinct m-ary tree configuration among the nodes.

Proof: Without any loss of generality, we shall outline the proof for $k = 3$. Following steps similar to that of Theorem 2, and using Theorem 3 and Lemmas 9–11, the nodes at various levels can be expressed as follows:

Level	Node
L_0	$R_2R_1R_0$
L_1	$R_2R_1''R_0$
L_2	$(R_1'' \oplus \hat{C}_1)R_1'R_0$
L_3	$(R_1' \oplus \hat{C}_1)R_1'((R_1'' \oplus \hat{C}_1 \oplus \hat{C}_2)$

where $R_1'' \in \boldsymbol{G}_{\boldsymbol{\alpha'}}(\boldsymbol{R}_1)\backslash\{R_1\}$ and $R_1' \in \boldsymbol{G}_{\boldsymbol{\alpha'}}(R_1)$. We choose to represent $\boldsymbol{G}_{\boldsymbol{\alpha'}}(R_1)$ as the set in which each member x consists of the variables of R_1 in the true and complementary form with the added constraint that $DEC(x) < m$.

Thus, the node R at L_0 has $m-1$ distinct predecessors. Each node at L_1 and L_2 has m predecessors. We now prove that any node at L_p is distinct form a node at L_q, $p \neq q$. In the above, $(R_1'' \oplus \hat{C}_1) \neq R_2$, because $R_1 \oplus \hat{C}_1 = R_2$ (Lemma 11) and $R_1'' \neq R_1$. Similarly, $((R_1'' \oplus \hat{C}_1) \oplus \hat{C}_2) \neq R_0$ because $R_2 \oplus \hat{C}_2 = R_0$ (Lemma 11) and we have proved that $R_1'' \oplus C_1 \neq R_2$. Hence the nodes at different levels are distinct. Furthermore, since $((R_1'' \oplus \hat{C}_1) \oplus \hat{C}_2) \neq R_0$, hence the nodes at L_3 do not have any predecessors and are the leaf nodes. This defines a $(k+1)$-level m-ary TC among the nodes. From Lemma 12, the m-ary TC is distinct for a distinct value of C. ■

Thus, the architecture can assume $m.2^{\lceil \log_2 m \rceil (k-1)}$ distinct TC's, one for each value of C. Fig. 13 depicts four of the twelve possible 3-ary TC's obtained with the architecture of Fig. 12. It can be verified that each TC can be synthesized from the root node, for any given value of the control code by following a procedure similar to Case 1.

V. SE Design Issues

In what follows, we discuss some issues in the design and implementation of the switching elements of the network.

When $m = 2$, each SE in S_{k-1} to S_1 (S_0) is a 2-by-2 (2-by-4) switch controlled by a single control line (see Fig. 6). Let us refer to the SE's of a 2-ary tree architecture as the *basic switching elements* (BSE's). When m is any other power of 2, each SE can be implemented using BSE's. Fig. 14(a) shows the scheme for implementing each SE in S_{k-1} to S_1 in the architecture for $m = 8$. In general, each SE is implemented with an α-stage $(s_{\alpha-1}, s_{\alpha-2}, \cdots, s_0)$ MIN, with each stage consisting of $m/2$ 2-by-2 BSE's. All the BSE's in a stage s_p, $p = \alpha-1, \alpha-2, \cdots, 0$, are controlled by the bit c_p of the code $c_{\alpha-1}c_{\alpha-2}\cdots c_0$ controlling the SE. The different stages are interconnected by a perfect shuffle connection. Furthermore, there is a perfect shuffle at the input of s_{k-1}. It can be verified that each input $i_{\alpha-1}i_{\alpha-2}\cdots i_0$ of the SE is connected to the output given by $i_{\alpha-1}i_{\alpha-2}\cdots i_0 \oplus c_{\alpha-1}c_{\alpha-2}\cdots c_0$, thus providing the 2^α switching states.

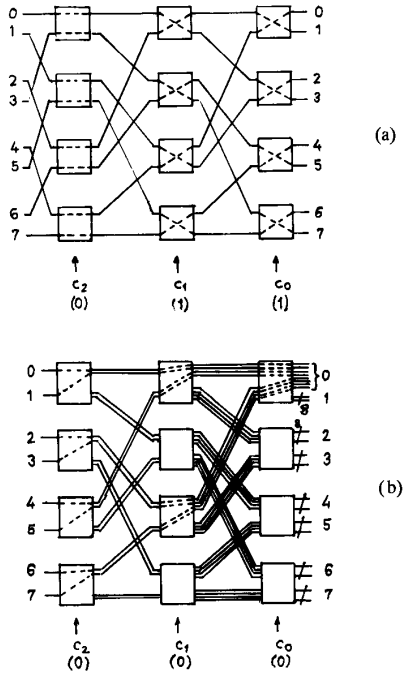


Fig. 14. Design of an SE using basic switching elements in the architecture with $m = 8$: (a) SE in stages S_{k-1} to S_1 and (b) SE in stage S_0.

Fig. 14(b) illustrates the implementation of each SE in S_0 for $m = 8$. The SE is again implemented as an α-stage shuffle network with $m/2$ switches in each stage but with no shuffle in front. The switches have switching states and construction similar to those of the 2-by-4 BSE's. The only difference is that each input and output line of the 2-by-4 BSE is replaced by $2^{\alpha-(p+1)}$ lines in each switch in stage s_p, $p = \alpha-1, \alpha-2, \cdots, 0$. This is required in order to obtain m^2 lines at the output of the SE.

When m is not a power of 2, a similar strategy can be adopted in the design of the SE's in S_0. The approach is to design each SE like an SE in the architecture for $2^{\lceil \log_2 m \rceil}$-ary tree structures with some of the lines being unused. For example, each SE in S_0 in the architecture for $m = 5$ can be designed similar to Fig. 14(b). However, an SE in S_{k-1} to S_1 requires a control logic along with a $\lceil \log_2 m \rceil$-stage shuffle MIN which is similar to Fig. 14(a). The control logic is necessary to detect the value of $(i_{\alpha-1}i_{\alpha-2}\cdots i_0 \oplus c_{\alpha-1}c_{\alpha-2}\cdots c_0)$ before realizing the output function of the SE.

VI. Concluding Remarks

We presented a generalized architecture for obtaining intra-topology reconfiguration in m-ary tree structures, where m is any integer > 1. The design is based on the generalization of the reconfigurable binary tree architecture proposed by us in [1]. The architecture for a reconfigurable m-ary tree structure with $N = m^k$ nodes (where k is any integer > 1) is implemented with a k-stage MIN. The basic principles of the design were evolved for the case when m is a power of 2. Subsequently, the design was modified for the case when m is an integer which is not a power of 2. In both cases, a single control code issued to the MIN achieves reconfiguration. A distinct m-ary TC is established among the nodes for each value of the control code. The architecture can assume m^k distinct TC's when m is a power of 2 and $m.2^{\lceil \log_2 m \rceil (k-1)}$ distinct TC's in general.

The reconfigurability of the architecture is based on the generalized reconfiguration equation. An analysis of this equation was done to prove that the architecture is capable of intra-topology reconfiguration. The results obtained while proving for reconfiguration are utilized to provide a procedure to synthesize the m-ary TC from the root node for any given control code. We also presented design considerations for efficient implementation of the switching elements of the MIN using BSE's.

The primary advantage of the proposed architecture is in the fast switching from one configuration to another. The reconfiguration is achieved by controlling the interconnection network and all the control code bits are broadcast simultaneously. Thus, the direct paths between the various nodes are established in parallel rather than stage by stage. Furthermore, conflict-free paths are established through the MIN and hence it is not necessary to check or acknowledge the data signals at each stage. These two factors make the system faster than existing methods which employ sequential switching in the MIN.

The architecture does not require any separate hardware in the nodes for achieving reconfiguration. Furthermore, there is a substantial reduction in the hardware of the MIN as compared to self-routing networks. When m is a power of 2, each SE in the architecture can be implemented as an α-stage MIN with BSE's. Each BSE is controlled by a 1-bit code which forms part of the control code C. Hence the logic for generating control signals in all the BSE's is eliminated. When m is not a power of 2, a hardware circuit is required in each SE in S_{k-1} to S_1 to detect whether each input has to be passed onto the α-stage MIN of BSE's. However, this circuitry will be simpler compared to the control logic in self-routing MIN's.

Furthermore, no algorithm is required to configure the nodes into TC's. Since each control code established direct paths between the nodes, no conflict resolution mechanism is required. These features result in the advantages of simple routing, low synchronization overhead, and fast reconfiguration.

References

[1] N. N. Biswas and S. Srinivas, "A reconfigurable tree architecture with multistage interconnection network," *IEEE Trans. Comput.*, vol. 39, pp. 1481–1485, Dec. 1990.

[2] S. P. Kartashev and S. I. Kartashev, "Analysis and synthesis of dynamic multicomputer networks that reconfigure into rings, trees, and stars," *IEEE Trans. Comput.*, vol. C-36, pp. 823–844, July 1987.

[3] C. L. Wu, T. Y. Feng, and M. C. Lin, "Star: A local network system for real-time management of imagery data," *IEEE Trans. Comput.*, vol. C-31, pp. 923–933, Oct. 1982.

[4] L. Snyder, "Introduction to the configurable, highly parallel computer," *IEEE Comput. Mag.*, vol. 15, pp. 47–56, Jan. 1982.

[5] J. Beetem, M. Denneau, and D. Weingarten, "The GF11 supercomputer," in *Proc. Annu. Symp. Comput. Architecture*, 1985, pp. 108–115.

[6] W. D. Hillis, *The Connection Machine*, Ph.D. dissertation, M.I.T. Press, 1985.

[7] D. K. Pradhan, "Dynamically restructurable fault-tolerant proecessor network architectures," *IEEE Trans. Comput.*, vol. C-34, pp. 434–447, May 1985.

[8] Z. Chen, C. Chang, and T. L. Chia, "RCAP—A reconfigurable cellular array processor for image processing," in *Proc. IEEE Conf. Comput. Architecture for Pattern Anal. and Image Database Mgt.*, 1985, pp. 80–86.

[9] J. Rothstein and A. Davis, "Bus automata, brains, and mental models," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, pp. 522–531, Apr. 1988.

[10] R. Miller, V. K. Prasanna Kumar, D. Reisis, and Q. F. Stout, "Meshes with reconfigurable buses," in *Proc. Fifth MIT Conf. Advanced Res. VLSI*, 1988, pp. 163–178.

[11] H. Li and M. Maresca, "Polymorphic-torus network," *IEEE Trans. Comput.*, vol. 38, pp. 1345–1351, Sept. 1989.

[12] A. L. Rosenberg, "The Diogenes approach to testable fault-tolerant VLSI processor arrays," *IEEE Trans. Comput.*, vol. C-32, pp. 21–27, Oct. 1983.

[13] D. Gordon, I. Koren, and G. M. Silberman, "Embedding tree structures in fault tolerant VLSI hexagonal arrays," *IEEE Trans. Comput.*, vol. C-33, pp. 104–108, Jan. 1984.

[14] C. S. Raghavendra, A. Avizienis, and M. Ercegovac, "Fault-tolerance in binary tree architectures," *IEEE Trans. Comput.*, vol. C-33, pp. 568–572, June 1984.

[15] J. A. B. Fortes and C. S. Raghavendra, "Dynamically reconfigurable fault-tolerant array processors," in *Proc. 14th Fault-Tolerant Comput. Symp.*, June 1984, pp. 386–392.

[16] R. Negrini, M. Sami, and R. Stefanelli, "Fault-tolerance techniques for array structures used in supercomputing," *IEEE Comput. Mag.*, pp. 78–87, Feb. 1986.

[17] A. S. M. Hassan and V. K. Agarwal, "A fault-tolerant modular architecture for binary trees," *IEEE Trans. Comput.*, vol. C-35, pp. 356–361, Apr. 1986.

[18] M. C. Howells and V. K. Agarwal, "Yield and reliability enhancement of large area binary tree architectures," in *Proc. 15th Annu. Symp. Fault-Tolerant Comput.*, June 1987, pp. 290–295.

[19] M. Lowrie and W. K. Fuchs, "Reconfigurable tree architectures using subtree oriented fault tolerance," *IEEE Trans. Comput.*, vol. C-36, pp. 1172–1182, Oct. 1987.

[20] A. D. Singh, "A reconfigurable modular fault-tolerant binary tree architecture," in *Proc. 15th Annu. Symp. Fault-Tolerant Comput.*, June 1987, pp. 298–303.

[21] A. D. Singh and H. Y. Youn, "A modular fault-tolerant binary tree architecture with short links," *IEEE Trans. Comput.*, vol. 40, pp. 882–890, July 1991.

[22] S. Dutt and J. P. Hayes, "On designing and reconfiguring k-tolerant tree architectures," *IEEE Trans. Comput.*, vol. C-39, pp. 490–503, Apr. 1990.

[23] K. P. Belkhale and P. Banerjee, "Reconfiguration strategies for VLSI processor arrays and trees using a modified diogenes approach," *IEEE Trans. Comput.*, vol. 41, pp. 83–96, Jan. 1992.

[24] J. F. Hayes, "A graph model for fault-tolerant computing systems," *IEEE Trans. Comput.*, vol. C-25, pp. 875–884, Sept. 1976.

[25] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comput.*, vol. C-30, pp. 934–947, Dec. 1981.

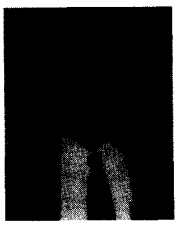
[26] J. C. Browne, "TRAC: An environment for parallel computing," in *Proc. 1984 IEEE COMPCON Spring*, pp. 294–298.

[27] P. Duclos *et al.*, "Image processing on a SIMD/SPMD architecture: Opsila," in *Proc. Ninth Int. Conf. Pattern Recognition*, 1988, pp. 430–433.

[28] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3) : Introduction and architecture," in *Proc. 1985 Int. Conf. Parallel Processing*, pp. 764–771.

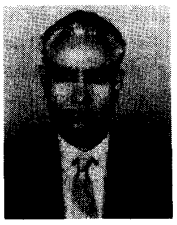
[29] S. Srinivas, "Dynamically reconfigurable architectures for supercomputing systems," Ph.D. dissertation, Indian Institute of Science, Dec.

1988.
[30] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. Palo Alto, CA: Morgan Kaufmann, 1992.
[31] A. M. Despain and D. A. Patterson, "X-tree: A tree structured multiprocessor computer architecture," in *Proc. Fifth Int. Symp. Comput. Architecture*, Apr. 1978, pp. 144–151
[32] A. K. Somani and V. K. Agarwal, "An efficient unsorted VLSI dictionary machine," *IEEE Trans. Comput.*, vol. C-34, pp. 841–852, Sept. 1985.
[33] E. H. Horowitz and A. Zorat, "The binary tree as an interconnection network : applications to multiprocessor systems and VLSI," *IEEE Trans. Comput.*, vol. C-30, pp. 247–253, Apr. 1981.
[34] J. L. Bentley and H. T. Kung, "A tree machine for searching problems," in *Proc. 1979 Int. Conf. Parallel Processing*, pp. 257–266.
[35] M. A. Bonucelli, E. Lodi, F. Luccio, P. Maestrini, and L. Pagli, "A VLSI tree machine for relational databases," in *Proc. 1983 Annu. Symp. Comput. Architecture*, pp. 67–73.
[36] Q. F. Stout, "Sorting, merging, selecting, and filtering on tree and pyramid machines," in *Proc. 1983 Int. Conf. Parallel Processing*, pp. 214–221.
[37] S. A. Browning, "The tree machine: A highly concurrent computing environment," Ph.D. dissertation, California Instit. of Technol., 1980.
[38] D. K. Hsiao, "The impact of interconnection network on parallel database computers," in *Database Machines and Knowledge Based Machines*, M. Kitsuregawa and H. Tanaka, Eds. New York: Kluwer Academic, pp. 216–224.
[39] S. W. Song, "A highly concurrent tree machine for database applications," in *Proc. 1980 Int. Conf. Parallel Processing*, pp. 259–268.
[40] S. J. Stolfo and D. P. Miranker, "DADO: A parallel processor for expert systems," in *Proc. 1984 Int. Conf. Parallel Processing*, pp. 74–82.
[41] D. J. DeWitt and D. Friedland, "Exploiting parallelism for the performance enhance of nonnumeric applications," in *Proc. AFIPS Conf.*, 1982, pp. 207–216.
[42] A. L. Gorin, A. Silberger, and L. Auslander, "Computing the 2-d DFT on the ASPEN parallel computer architecture," in *Proc. 1987 Int. Conf. Parallel Processing*, pp. 921–923.
[43] D. E. Shaw, "SIMD and MSIMD variants of the NON-VON supercomputer," in *Proc. IEEE COMPCON Spring 1984*, pp. 360–363.
[44] *DBC/102 Database Computer Concepts and Facilities*, Teradata Corp. document, 1983.
[45] B. W. Arden and H. Lee, "A multi-tree structured network," in *Proc. COMPCON 78*, Sept. 1978, pp. 201–210.
[46] J. R. Goodman and C. H. Sequin, "Hypertree: A multiprocessor interconnection topology," *IEEE Trans. Comput.*, vol. C-30, pp. 923–933, Dec. 1981.
[47] B. L. Menezes and R. Jenevein, " The KYKLOS multicomputer network: Interconnection strategies, properties, and applications," *IEEE Trans. Comput.*, vol. 40, pp. 693–705, June 1991.
[48] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, pp. 1145–1155, Dec. 1975.
[49] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Comput.*, vol. C-30, pp. 771–780, Oct. 1981.
[50] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.



Sampalli Srinivas (M'92) was born in Bangalore, India, on March 31, 1963. He received the B.E. degree in electronics engineering from the Banglore University and the Ph.D. degree from the Indian Institute of Science, Bangalore, in 1985 and 1989, respectively. His Ph.D. dissertation was in the area of reconfigurable parallel architectures for computing systems.

From July 1989 to October 1990, he was a Member Technical Staff at the Centre for Development of Advanced Computing (C-DAC), Bangalore. During this period, his work was mainly concerned with the design of parallel architectures and the design and analysis of parallel algorithms on transputer networks. From October 1990 to August 1992, he was a Post-Doctoral Fellow at the Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada. Since September 1992, he has been working as an Assistant Professor in Computing Science at Dalhousie University, Halifax, NS, Canada. His current research interests are in the areas of parallel and distributed computing, with emphasis on reconfigurable parallel architectures, fault-tolerant computing, parallel algorithms, and compiler-directed approaches for the latency problem in multiprocessors.



Nripendra N. Biswas (M'58–SM'65) B.Sc. (Hons.) (Calcutta), B.S. and M.S. (Indian Institute of Science, Bangalore), and Ph.D. (Indian Institute of Technology, Kharagpur) has served as Professor at the University of Roorkee, St. Louis University, and the Indian Institute of Science. He has also been a Visiting Professor/Scientist at AT&T Bell Laboratories, Murray Hill, Syracuse University, the University of Southwestern Louisiana, and Stanford University.

After he retired from the Indian Institute of Science, he became an Emeritus Professor at the Institute by a grant from the Council of Scientific and Industrial Research, New Delhi, India. He is currently engaged in research in the area of Boolean neural networks. He has published more than 80 papers, and authored six books.

Dr. Biswas is a Fellow of the Indian National Science Academy.