On Implementing Large Binary Tree Architectures in VLSI and WSI

HEE YONG YOUN, MEMBER, IEEE, AND ADIT D. SINGH, MEMBER, IEEE

Abstract—**The complete binary tree is known to support the parallel execution of important algorithms, which has given rise to much interest in implementing such architectures in VLSI and WSI. For large trees, the classical H-tree layout approach suffers from area inefficiency and long interconnects. Other proposed schemes are not well suited for the implementation of defect-tolerant designs. This paper presents an efficient scheme for the layout of large binary tree architectures by embedding the complete binary tree in a two-dimensional array of processing elements. Our scheme utilizes virtually 100 percent of the processing elements in the array as computing elements; it also shows substantial improvements in propagation delay and maximum edge length over H-tree layouts. We show that our layouts readily lend themselves to fault-tolerant designs for overcoming fabrication defects in large area and wafer scale implementations of binary tree architectures.**

Index Terms—**Area efficiency, embedding, layout, maximum edge length, propagation delay, trees, VLSI, WLSI.**

I. Introduction

THE complete binary tree architecture has long been investigated for distributed and parallel processing in applications such as dictionary and database machines [1]–[3]. With recent advances in VLSI, several researchers have proposed implementing entire binary tree architectures on a single piece of silicon. For conventional size dies, this is practical for trees with perhaps as many as a hundred nodes, if the processing elements in the nodes are relatively small. Larger trees, or even relatively small trees with large nodes will, however, require significantly more silicon area than can be fabricated defect-free with viable yield. Such large area (even full wafer) circuits are expected to employ extensive on-chip fault tolerance capabilities [18]–[22] to overcome the effects of these defects.

Early proposals for laying out tree architectures in VLSI were based on the classical H-tree approach [9]. Such layouts, however, exhibit area inefficiencies and long interconnections, particularly for large trees with large processing nodes. More recent designs [8], [10], [11] employ tile-based schemes to improve silicon utilization and propagation delay. These schemes have actually been developed for embedding binary trees in dynamically reconfigurable mesh and hexagonal arrays. As such, they incorporate a switching capability in each processing node to support the reconfiguration. However, for designs that are small enough in area for a viable defect-free yield, some of the same schemes can also be used for laying out a fixed binary tree architecture. The scheme for embedding trees in hexagonal arrays presented in [11] appears particularly attractive for such an application since it achieves virtually 100 percent utilization of the processing elements. Of course, any unused interconnections in the hex-connected host topology, as well as the reconfiguration switches, need not be implemented in a static tree design.

Note, that in these tile-based designs, some processing elements also act as connectors. In a static layout, this would imply a connection through the processing element. While this can be easily achieved, in principle, with multilayer interconnects, it is generally not desirable in a fault-tolerant design for the following reasons. 1) Interconnections layed out over active components and lower level interconnect layers are more susceptible to faults because they run over an uneven surface. Since many fault-tolerant schemes are less effective at handling interconnect failures as compared to processor failures, they often require that interconnects be conservatively layed out in special tracks or channels, without any other circuit component below them. 2) Interconnects implemented in a high-level metal layer are usually made much wider (typically 3–7 times miniminum the feature size) to overcome the increased possibility of a break in the line. This results in enhanced capacitance on the lines and degraded performance. Capacitive loading is a particular concern for long interconnects between nonadjacent nodes which exist when a node acts as a connector. 3) Several important static restructuring technologies for defect tolerance require channel-based interconnects. For example, laser welding [24] requires that there be no active components under the weld locations. This implies that the interconnections must be layed out separate from the processing nodes.

In view of the above, it is important to develop efficient layouts for binary trees that employ tracks or channels between the processors for running the interconnects. No connections must run over processing elements (or other active circuitry). Such a layout model has been called a wafer scale model in [23]. It is clear that the layout approach based on embedding the tree in a hex-connected array [11] discussed above is not efficient under such a model. For hexagonal or

Manuscript received July 6, 1988; revised November 22, 1988. This work was supported in part by the National Science Foundation under Grant MIP-8808325.

H. Y. Youn was with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003. He is now with the Department of Computer Sciences, University of North Texas, Denton, TX 76203.

A. D. Singh is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003.

IEEE Log Number 8826378.

even square layout for the processors, the tracks cannot run in a straight line, thus resulting in longer interconnects and increased total interconnect area. Furthermore, if we fix the I/O port locations consistently on the edges of each processor, more than one track is needed in many of the channels. This increased area also increases the circuits' susceptibility to interconnection failures.

In this paper, we present a new tile (module) based design for laying out the complete binary tree by embedding it in a two-dimensional host grid of processors with channel-based interconnections. For any size of tree, we show our design to be better than the direct extension of either the H-tree layout or the schemes in Gordon [11] to the wafer scale model discussed above, in both area efficiency and maximum edge length. Our proposed layouts achieve nearly 100 percent area utilization and also short propagation delay and internode links for practical size trees. We show that they also readily lend themselves to fault-tolerant designs for overcoming fabrication defects in large area and wafer scale implementations.

The rest of this paper is organized as follows. Section II presents our new tree layout scheme. In Section III, we evaluate our design on area efficiency, propagation delay, and maximum edge length. In Section IV, we present a modified layout that reduces the maximum edge lengths for larger trees at a small expense of additional layout area. Our results are compared to other designs in Section V. In Section VI, we discuss how our layout approach can incorporate modular fault tolerance schemes to allow large area and even full wafter implementation. We conclude in Section VII.

II. Proposed Tree Layout Scheme

In this section, we present an area optimal planar layout scheme for binary trees by considering the embedding of the binary tree topology in a two-dimensional rectangular array of processors. Note that the desired computational interconnection is assumed to be a *complete binary tree*. Furthermore, as in other proposed designs, we assume that the processors are relatively large with edge size much bigger than the width of the interconnect. Therefore, the main concern of our design is the high utilization of processors; and for performance, short internode and root-to-leaf distances. Later, in Section V, we also discuss the effect of relatively wide interconnects on our layouts.

A. Embedding Strategy and Structure of Basic Modules

The proposed tree layout scheme uses a heirarchical strategy such that any required size of tree larger than four levels is layed out by connecting an appropriate *number* and *type* of basic modules. Each basic module is a 4 × 4 square array of processors and contains a four-level tree (Fig. 2). Thus, 15 out of the 16 processors in the basic module are utilized as tree nodes. The remaining unused processor in each basic module is used as a tree node at some higher level, when the basic modules are connected together to build a larger tree. (This strategy is also employed in the hexagonal array design in [11].) In other words, each basic module contains a four-level leaf subtree of the overall tree and one additional node belonging to a higher level in the tree. See Figs. 3–5 for some examples.

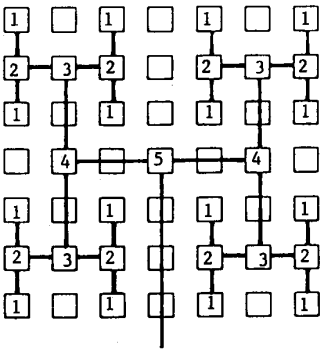


Fig. 1. A five-level tree embedded in the H-tree layout.

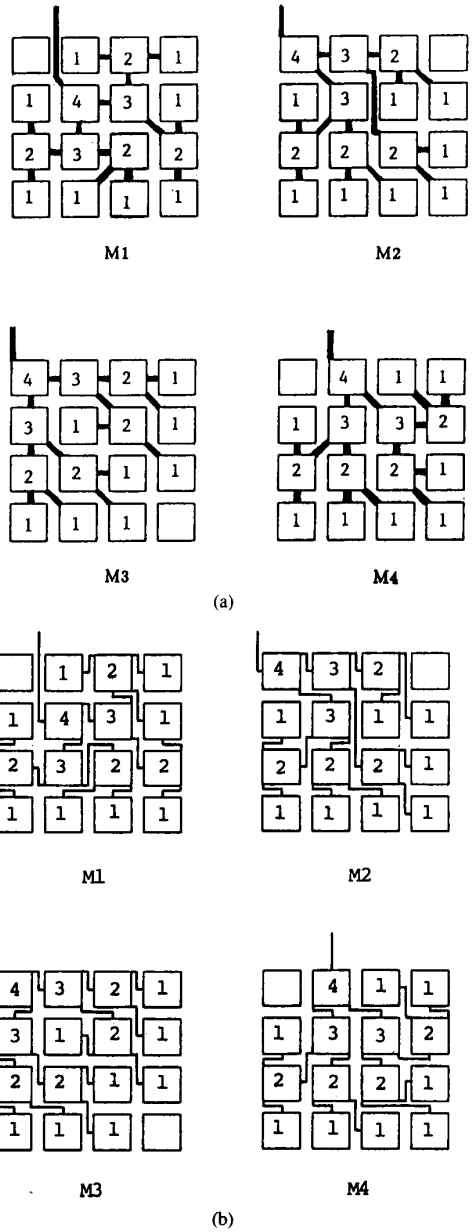


Fig. 2. Four types of basic module. (a) Schematic showing connections. (b) Implementation using channel interconnects.

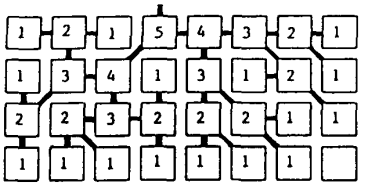


Fig. 3. A five-level tree embedding using two basic modules.

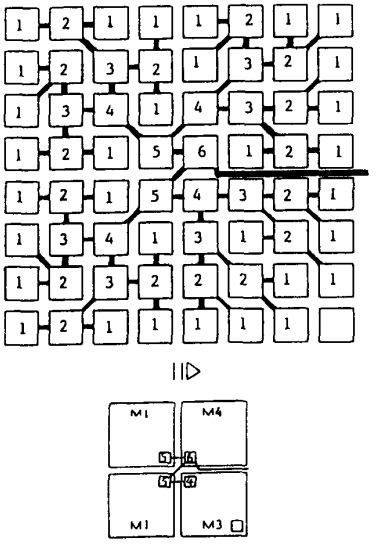


Fig. 4. A six-level tree embedding using four basic modules.

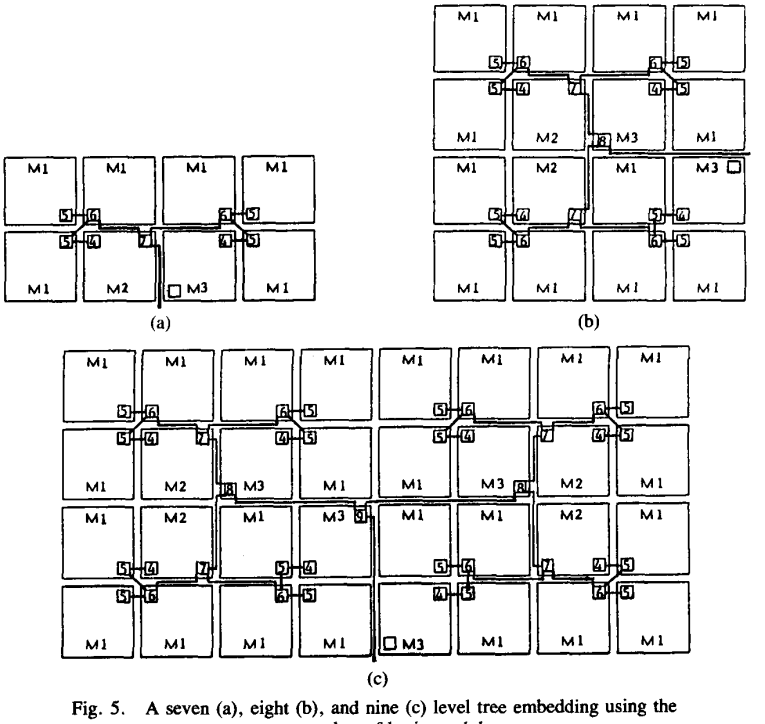


Fig. 5. A seven (a), eight (b), and nine (c) level tree embedding using the proper number of basic modules.

Whenever a larger tree is constructed using the basic modules, the root node of the tree should be located at the center of the two-dimensional host array for minimum propagation delay. In order to realize this, we employ four types of 16 processor basic modules. These are different from one another in the relative position of the unused node with respect to the root node for the four-level subtree in each basic module. Fig. 2(a) shows the four types of basic modules. We refer to them as $M1$, $M2$, $M3$, and $M4$, respectively. The number in each box (processor) denotes the level of the node in the tree, with the leaf nodes being level 1. Observe that all nodes at adjacent levels, except for two nodes in basic modules $M2$, are physical neighbors and can be connected with short links. This allows us to construct larger trees using these basic modules which are not only area efficient but also have short propagation delay.

Note that the diagonal connections shown in Fig. 2(a) are for illustration only. In practice, under our channel model, these would be implemented using horizontal and vertical tracks in a "staircase" manner. Fig. 2(b) shows how this can be achieved using only a single track in the channel. The I/O ports for each node are fixed with the input being applied to the middle of an edge, and the two output ports being located at the two opposite corners. We do, however, assume that the tree nodes are large enough so that the output ports can provide a connection through either edge adjacent to the node corner without incurring a significant area penalty.

B. Five- and Six-Level Trees

A five-level tree can be embedded by laying out and connecting one $M1$ basic module with one $M3$ basic module side by side as shown in Fig. 3. Observe that the unused node in the basic module $M1$ is utilized as the root node of the five-level tree. Actually, one $M1$ basic module with either an $M1$ or $M2$ basic module can also be used to construct a five-level tree. However, these combinations are not preferred because, as can be easily seen, they lead to longer propagation delay.

Fig. 4 shows how a six-level tree can be constructed using two $M1$, one $M3$, and one $M4$ basic module. Here the number at one corner node of each basic module indicates the level of the node in the embedded tree. Also, the empty small box indicates the unused node. Other combinations of four basic modules such as four $M1$, or three $M1$ and one $M3$ can also realize the six-level tree. Again these combinations are not preferred due to longer propagation delay. The basic module $M4$ is used only once in the six-level tree embedding for achieving minimum propagation delay.

C. Trees Larger than Six Level

A seven-level tree is shown in Fig. 5(a). Notice that the unused processor is located in the middle of an edge and adjacent to the channel where the interconnection from the root node emerges from the array.

Trees of arbitrary size larger than seven levels can be obtained by repeatedly taking a tree of given size and obtaining a tree with one more level. At each level, we ensure that the unused processor is in the middle of the edge and adjacent to the connection to the root node. The procedure to obtain an $n + 1$ level tree from an n level tree is as follows.

Obtain a mirror image layout of the n level tree along the edge containing the unused processor. Combine the two layouts and locate the root of the $n + 1$ level tree in the unused processor in the original n level tree. Let the connection from the root node of the new $n + 1$ level tree to the outside run in the channel between the new root and the unused processor. Let this connection emerge from the array in the middle of the edge closer to the root node.

Finally, we must move the unused processor to the outside edge. To do this, locate the set of modules containing the $n - 2$ level subtree that includes the four-level subtree in the module with the unused processor. Rotate this set of modules (in their correct relative position) about an axis parallel to the appropriate array edge so as to move the unused processor to the outside of the array.

We now have an $n + 1$ level tree with the unused processor on an outside edge and adjacent to the connection of the root node. This process can be repeated recursively, beginning with the seven-level tree, until a tree of desired size is obtained.

Trees of arbitrary size larger than seven level can be obtained by repeatedly taking two trees of a given size and constructing a larger tree with one additional level. This process can be repeated recursively beginning with two six-level trees until a tree of the desired size is obtained. Fig. 5(a), (b), and (c) shows how trees of seven, eight, and nine level are embedded using this approach. Observe that in each case the complete tree is composed of two subtrees that are one level smaller. Also, the root node always resides at center in the host processor array. This ensures that the root-to-leaf propagation delay in the embedded tree is as short as possible, as we prove in the next section.

We next evaluate our layout scheme on area efficiency, propagation delay, and maximum edge length. In Section V, we shall compare it to designs that result from directly extending the tile-based scheme in Gordon [11] to a grid model with channel interconnects.

III. Evaluation of Proposed Tree Layout Scheme

A. Evaluation Criteria

For the evaluation of the effectiveness of an embedding scheme to be used to lay out large trees, the following three criteria are important.

- Chip area efficiency—defined by the ratio of the active chip area utilized for actual computations to the entire chip area. When each processing element is relatively large, almost all the chip area is taken up by processing elements. The area efficiency is then the ratio of the number of processing elements actually utilized as tree nodes to the total number of processing elements implemented on the chip. Because the cost of a VLSI circuit is greatly influenced by the chip area, this factor evaluates the cost effectiveness of the embedding scheme.
- Propagation delay (PD)—defined by the maximum data propagation time between the root node and the leaf nodes. This determines the setup time and data latency when the tree architecture executes a problem in parallel.

• Maximum edge length (MEL) [6]—defined by the maximum edge length between any pair of two directly connecting nodes. This is important for two reasons. If the entire processor array operates with a common synchronization clock, then the maximum clock rate is determined by the propagation delay in the longest tree edge. This factor limits the execution speed of the design. Also, in many asynchronous binary tree architectures the frequency of communications between nodes at two adjacent levels doubles as we move up the tree, from the leaf to the root node. Therefore, the bandwidth of upper edges is often required to be very high for efficient system operation. Thus, it is important that the edge lengths at the higher levels in the tree be short.

B. Area Efficiency

As mentioned in the previous subsection, the area efficiency of an embedding scheme has traditionally been estimated by ratio of the number of utilized processors to the total number of processors in the host array in which the tree topology is embedded. It is obvious from Figs. 3–5 that *only one processor is left unutilized in our design for any size of tree.* Thus, the area efficiency of our scheme is $(2^k - 1)/2^k$ for a $k(\geq 4)$ level tree. This quickly converges to 100 percent as the size of the embedded tree grows large. Because the number of nodes in a tree of any size is always is $2^k - 1$ (always odd) and we embed it in an array with 2^k processors, having one unused processor is unavoidable. Therefore, our tree embedding scheme is optimal with respect to utilizing the processors. Note that the area of interconnection buses running between the modules is not reflected in the above area efficiency consideration because the interconnection area is assumed to be small in comparison to the processor area.

C. Propagation Delay

The propagation delay (PD) in a tree architecture can be estimated by the maximum distance between root and leaf nodes as in the linear propagation delay model assumed in [11] and [17]. This can be expressed in terms of the processor edge dimension, which is the length of one side of a processor, assumed to be a unit square. To obtain a uniform and consistent measure of interconnect distances independent of details of the layout, we index the processor in row i and column j as (i, j). We also assume that all data paths run horizontally and vertically, i.e., Manhattan interconnections. Then the distance between two directly connected processors (i_1, j_1) and (i_2, j_2) is the sum of the differences between i_1 and i_2, and j_1 and j_2. The maximum root-to-leaf distance can be found by comparing the distances of all paths from the root node to the leaf nodes. These can be obtained by adding up the distances of each successive pair of two directly connected processors on the path from the root node to the leaf node. For example, the maximum distance of five-level tree in Fig. 3 is 6 because the distance from the root node (1, 4) to leaf node (4, 7) via (1, 5), (2, 5), and (3, 6) is 1 + 1 + 2 + 2 = 6.

We now prove a lower bound for the optimal (minimal) propagation delay that can be obtained using this approach for any efficient embedding of a binary tree in a rectangular array of processors with channel-based interconnections.

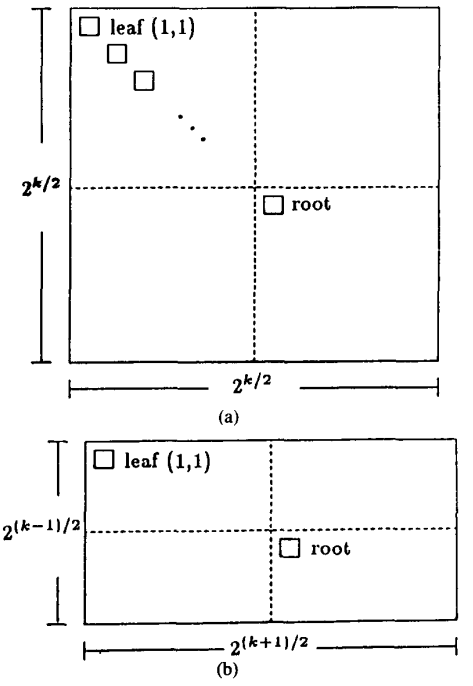


Fig. 6. Embedding of a k-level tree with optimal propagation delay. (a) Even k, root $(2^{(k-2)/2} + 1, 2^{(k-2)/2} + 1)$. (b) Odd k, root $(2^{(k-3)/2} + 1, 2^{(k-1)/2} + 1)$.

Theorem 3.1: A lower bound for the optimal porpagation delay ($PD_o(k)$) for a k level tree ($2^k - 1$ nodes) embedding on a rectangular array with 2^k processors is $2^{k/2}$ and $3(2)^{-1.5}2^{k/2}$ when k is *even* and *odd*, respectively.

Proof:

Case 1: k is even.

Assume that a k level tree is embedded with optimal propagation delay in a two-dimensional rectangular processor array with 2^k nodes. Then the optimal (minimal) propagation delay can be obtained only when the root node is located at one of the four processors at the center of the array *and* at least three of the four nodes (processors) located at the four corners are utilized as leaf nodes, as shown in Fig. 6(a). If both these conditions are not satisfied, then the maximum root-to-leaf distance is always greater than the theoretical optimal distance because the distance between the root node and the node at one of the four corners (which is not the leaf node) is already larger than the optimal value $PD_o(k)$. Also, it can be easily seen that the diagonal length of a rectangular array of fixed area (number of nodes) is minimal when it has sides of equal length. Therefore, the optimal propagation delay for the k level tree embedding (k even) is achieved when all the four sides of the array have $2^{k/2}$ nodes. This distance between the root node $(2^{(k-2)/2} + 1, 2^{(k-2)/2} + 1)$ and the leaf node (1, 1) is $2^{k/2}$.

Case 2: k is odd.

When the level of the embedded tree is odd, the host array with 2^k processors cannot have equal sides. Since for the shortest diagonal, the sides of a rectangular array must be as close in length as possible, the optimum propagation delay is achieved when the number of nodes on vertical and horizontal

sides of the array is $2^{(k-1)/2}$ and $2^{(k+1)/2}$, respectively, as shown in Fig. 6(b). Then the distance between the root node $(2^{(k-3)/2} + 1, 2^{(k-1)/2} + 1)$ and the leaf node $(1, 1)$ is $2^{(k-3)/2} + 2^{(k-1)/2} = 3(2)^{-1.5}2^{k/2}$. □

Note that Theorem 3.1 is based on geometric arguments similar to those used to obtain a lower bound in [11]. It is valid only under our layout model along with the assumption that the overall layout is rectangular and utilizes all but one node position in the array. If this efficiency requirement is relaxed, or if the overall array is allowed to have irregular edges, then the propagation delay can be further improved as, for example, in Fig. 12(a) in [11]. However, even with the relaxed efficiency considerations, no systematic way for generating such designs for arbitrary size trees has yet been presented.

The lower bound for the optimal maximum edge length can be obtained using Lemma 3.1.

Lemma 3.1: A lower bound for the optimal maximum edge length ($\mathrm{MEL}_o(k)$) for a k level tree embedding on a rectangular array with 2^k processors is $\lceil (O_P D(k))/(k-1) \rceil$.

From Figs. 3–5 the following equations for the propagation delay ($\mathrm{PD}(k)$) in our scheme for a k level tree embedding can be obtained based on the approach just discussed.

$$\mathrm{PD}(k) = \begin{cases} 2k-4 & \text{if } k \le 6 \\ 2^{k/2}+k-6 & \text{if } k>6 \text{ and even} \\ 3(2)^{-1.5}2^{k/2}+k-6 & \text{if } k>6 \text{ and odd.} \end{cases} \quad (1)$$

The propagation delay of our design for tree sizes up to six levels is same as the lower bound for the optimal propagation delay in our model. For trees of larger sizes, it is $k - 6$ larger than the lower bound. Note, however, that we have derived a lower bound purely on geometric arguments. It is not at all clear that this lower bound can be realized in practice. Although our interest remains primarily in practical sized designs, it is still interesting to observe that since the $k - 6$ term increases only linearly while the $2^{k/2}$ term increases geometrically as k increases, the propagation delay of our scheme is asymptotically optimal.

D. Maximum Edge Length

The maximum edge length ($\mathrm{MEL}(k)$) of our design is readily seen from the figures to be

$$\mathrm{MEL}(k) = \begin{cases} 2 & \text{if } k \le 6. \\ 2^{\lfloor (k-7)/2 \rfloor + 2} + 1 & \text{if } k \ge 7. \end{cases} \quad (2)$$

For small size trees ($k \le 6$) our tree embedding scheme displays optimal maximum edge length because all the tree nodes are either adjacent or diagonal neighbors. By Lemma 3.1, the optimal maximum edge length for a six-level tree embedding is 2; this is same as that of our scheme. However, for larger trees, the H-tree pattern interconnections of nodes at levels higher than six can result in relatively long upper level edges, and degrade performance. Thus, we still need to develop a design which reduces the maximum edge length for trees with more than six levels, since such large designs now appear viable with WSI technology.

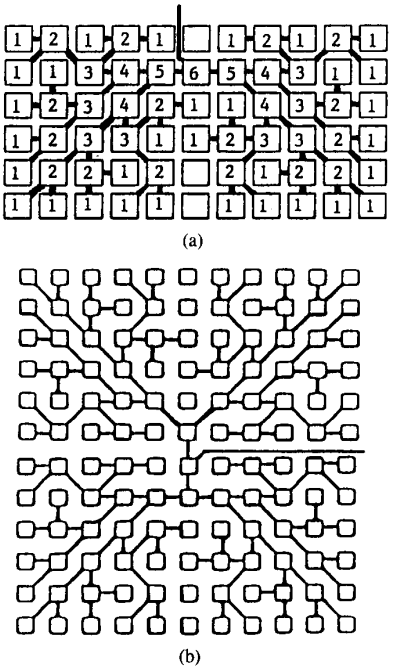


Fig. 7. A six (a) and seven (b) level tree embedding with optimal maximum edge length.

Recall from the discussion in Section III-C that the propagation delay for our design is near optimal for any tree size. Since this is the root-to-leaf node distance, the maximum edge length can be minimized by equally dividing this distance among the edges along each path from the root to a leaf node. In the next section, we represent a design for achieving this for practical size trees at a small cost in layout area. We then also discuss the (asymptotically) optimal minimax edge length results of Ruzzo and Snyder [25].

IV. Minimizing the Maximum Edge Length

A. Design Using Larger Basic Modules

In general, a k level tree can be constructed by laying out two $k - 1$ level trees side by side with the root node at center of the whole array. Fig. 7(a) shows a basic module containing a six-level complete binary tree embedded in a 6×11 processor array with the root node at the center of one side. When two such modules are combined to realize a seven-level tree as in Fig. 7(b), 96.2 percent area efficiency can be achieved with optimal maximum edge length (two), because all the tree nodes are physical neighbors. Note that the layout in Fig. 7(a) is a "custom" layout and was not obtained from smaller modular building blocks. An obvious problem in extending this approach to larger trees is the difficulty of developing such custom layouts for arbitrary size trees while ensuring high-area efficiency and short maximum edge length.

B. Design Using Four-Level Basic Modules

We now present a systematic scheme for reducing the maximum edge length in the layout approach based on the four-level basic modules presented in Section II. Our design achieves this at the cost of a small increase by requiring an

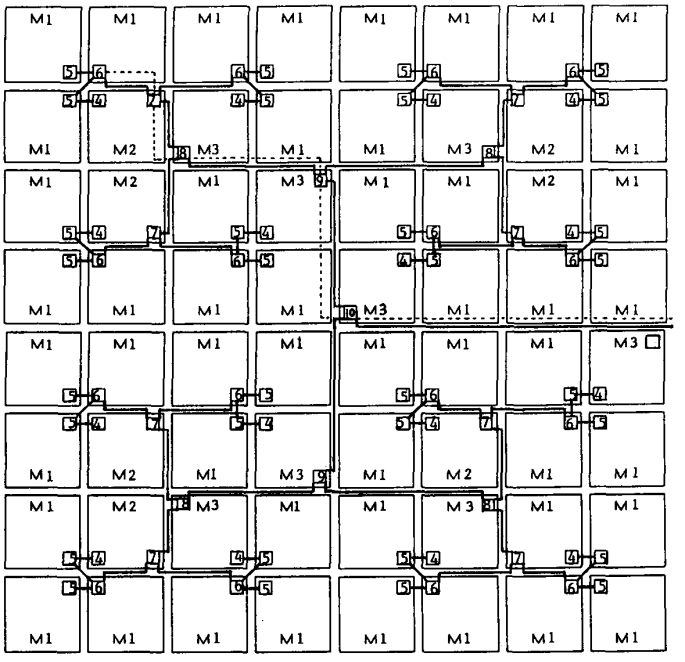


Fig. 8. A ten-level tree embedding without minimizing the maximum edge length.

extra bus between each four-level basic module. Recall that in any design based on our earlier layout scheme, each basic module (except one) contains a four-level subtree and exactly one additional node at a higher level (>4) in the tree. This allows our scheme to achieve 100 percent area efficiency. For example observe, in the nine-level tree in Fig. 5(c), that each basic module contains exactly one tree node at some level from 5 to 9. Also notice that the longest edges are those between the root and its two children. This is due to the H-tree pattern interconnection among the basic modules, and the fact that the higher level (>4) node is located only at one corner of any basic module. The maximum edge length can be reduced if the higher level node in each basic module can be relocated to other positions so as to equalize the higher level edges in the overall tree. This requires additional types of basic modules with different positions of the higher level node with respect to the root of the four-level subtree in the modules.

To explain our proposed design, let us assume that a ten-level tree is required to be layed out. Following the approach presented in Section II-C, the tree can be embedded in a 32×32 array as shown in Fig. 8. Also indicated in Fig. 8 is the general H-tree pattern of interconnections between the basic modules. Notice the long edge connection between the root of the tree (marked 10) and the root of the two nine-level subtrees. This maximum edge length can be shortened by equalizing the distances between successive level nodes along the H-tree pattern as shown in Fig. 9.

To understand the procedure used to obtain the layout in Fig. 9, observe that since the unused location in each basic module is required to house a higher level node (for area efficiency), the root of two five-level subtrees must be located at (4, 4) and (5, 4) in the layout. The root node (level 10) is located at (16, 17) in our example. (It obviously must be located at one of the four center positions.) Thus, along the path shown in dotted in Fig. 8, the total length of the tree edges from level 5 node (5, 4) to the level 10 root node is 26. By relocating the nodes at levels 6–9 in Fig. 8, we attempt to make each edge connecting successive levels in the tree of length $\lceil 26/5 \rceil = 6$ (or less). This is the optimal maximum edge length between level 5 nodes and the root node. The relocation requires additional four-level basic modules, shown in Fig. 10, displaying all the other boundary locations of the unused node. Also, the unused node may have to be exchanged repeatedly between physically adjacent modules until the unused node is propogated to the site where a higher level node is to be located. It is clear that this can always be done with planar interconnections since there are no other connections perpendicular to the edges of the H-tree. Any additional connections required between adjacent modules to affect the exchange will not make the layout nonplanar. However, even with additional basic modules, it is not always possible to completely equalize the edges. This is because a higher level node can only be relocated along its arm of the H-tree pattern. For example, node 6 along the path shown dotted can only be located at (4, 5), (4, 6), (4, 7), or (4, 8).

We can now state a procedure that can be used to obtain the layouts with reduced maximum edge length.

Procedure 4.1:

1. Obtain a layout as discussed in Section II-C.
2. Starting with the root node, construct the H-tree interconnect data path to the level 5 node in the module farthest from the root node, as shown in dotted in Fig. 8.
3. Begin with the five-level node farthest from the root node.

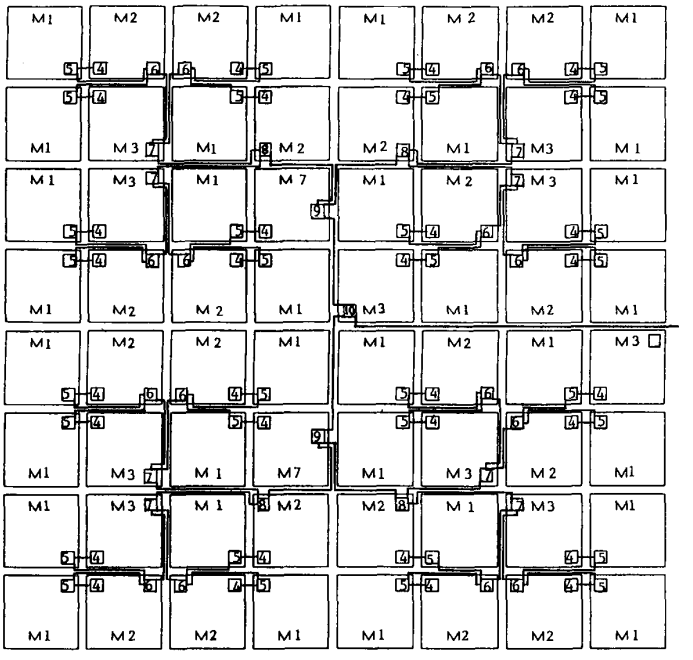


Fig. 9. A ten-level tree embedding with minimizing the maximum edge length.

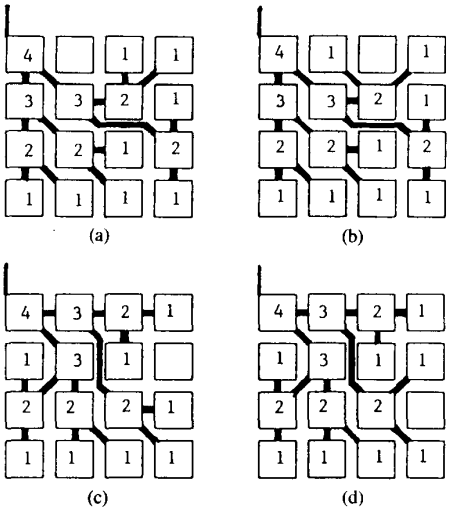


Fig. 10. Additional four-level basic modules for minimizing the maximum edge length. (a) $M5$. (b) $M6$. (c) $M7$. (d) $M8$.

4. Compute the optimal maximum edge length from the node under consideration to the root node.
5. Locate the next higher level node as far as possible along the H-tree path from the node presently being considered, but no further than the maximum optimum edge length, subject to the restriction that it remain in the same arm of the H-tree.
6. If this new node is a child of the root node, then stop. Else go back to step 4.
7. From symmetry reasoning, place all other higher level nodes and select appropriate basic modules for each location from the complete set of basic modules in Figs. 2 and 10.

It can be easily confirmed that the above procedure generates the layout in Fig. 9.

The restriction that a high-level node can only be relocated along its arm of the H-tree restricts the effectiveness of Procedure 4.1 to, at best, reducing the maximum edge length by half for large trees. For practical designs, this is a useful improvement since propagation delays for long interconnects in WSI are expected to be quadratic in length because of wire resistance, rather than the capacitive linear delays observed in short lines. Note, however, that the maximum edge length for our design remains $O(\sqrt{n})$(as for the basic H-tree). Ruzzo and Snyder [25] have presented a layout scheme that asymptotically achieves the $O(\sqrt{n}/\log\ n)$ optimum value for the maximum edge length. However, their design is inefficient in that it has many unused nodes in the array. Also, for practical size trees of up to at most a few thousand nodes, the reduction in maximum edge length compared to the design proposed here is unlikely to be significant due to the inefficiencies introduced by the unused nodes.

V. Comparisons to Other Designs

The overall effectiveness of a tree layout scheme can be effectively evaluated by three critera—area efficiency, propagation delay, and maximum edge length—as discussed in Section III. In this section, our new proposed scheme is compared to two other approaches—the classical H-tree scheme [9] and a direct extension of the scheme for mesh-connected arrays in Gordon [11] to our layout model. In viewing the comparisons, it is important to recognize that

Gordon's work is aimed at dynamically reconfigurable arrays and is not optimized for our static layout model. However, it is still worthwhile to consider it here because it is a highly effective and efficient design; it is not immediately obvious that its extension to our layout model will not outperform any new scheme that we may devise.

First we briefly introduce these other schemes.

A. H-tree Scheme

The H-tree scheme [9] embeds trees in the "H" pattern. A five-level tree embedded in a 7 × 7 square array using this scheme is shown in Fig. 1. Observe that a significant number of processors are used only as connecting elements to route signals between nodes. Let $A_h(k)$ be the number of processors required for embedding a k level tree, and $PD_h(k)$ be the propagation delay. Also let $MEL_h(k)$ be the maximum edge length. Then from [26] we have

$$A_h(k)=\begin{cases}(2^{(k+1)/2}-1)\times(2^{(k+1)/2}-1) & \text{if } k \text{ is odd}\\ (2^{(k+2)/2}-1)\times(2^{k/2}-1) & \text{if } k \text{ is even}\end{cases} \quad (3)$$

$$PD_h(k)=\begin{cases}2^{(k+1)/2}-2 & \text{if } k \text{ is odd}\\ 3\times 2^{(k-2)/2}-2 & \text{if } k \text{ is even}\end{cases} \quad (4)$$

$$MEL_h(k)=\begin{cases}1 & \text{if } k<4\\ 2^{\lfloor k/2\rfloor-1} & \text{if } k\geq 4.\end{cases} \quad (5)$$

The area efficiency for the H-tree scheme is $(2^k - 1)/A_h(k)$. Fig. 12 shows that the area efficiency converges to 50 percent as the level of the embedded tree grows large. Figs. 13 and 14 show the changes in propagation delay and maximum edge length for 4–10 level trees.

B. Extension of Gordon's Scheme

Gordon [11] has recently proposed a hierarchical tree embedding scheme which improves the efficiency of previous work [10]. We have already explained in the Introduction that the hex-connected schemes are not suited to our layout model. For mesh-connected arrays, in [11] he has proposed five basic tiles such that each tile embeds a six-level tree on a 9 × 9 rectangular array. Each basic tile is designed to contain many (almost half) unused nodes on their four sides in order that the basic tiles can be interlocked side by side when they are put together to build a larger tree. Fig. 11 shows one of his basic tiles. If we define $A_G(k)$, $PD_G(k)$, and $MEL_G(k)$ as we did for the H-tree scheme, then from [11]

$$A_G(k)=\begin{cases}(2^{(k+1)/2}+1)\times(2^{(k-1)/2}-1) & \text{if } k \text{ is odd}\\ (2^{k/2}+1)^2 & \text{if } k \text{ is even}\end{cases} \quad (6)$$

$$PD_G(k)=\begin{cases}2^{k/2} & \text{if } k>6 \text{ and even}\\ (1.06066)2^{k/2} & \text{if } k>6 \text{ and odd}\end{cases} \quad (7)$$

$$MEL_G(k)=\begin{cases}3 & \text{if } 5<k<9\\ 2^{\lfloor k/2\rfloor-1} & \text{if } k\geq 9.\end{cases} \quad (8)$$

C. Comparison of Area Efficiency

From Fig. 12, we observe that for any size of tree, the proposed scheme is the most area efficient among these three

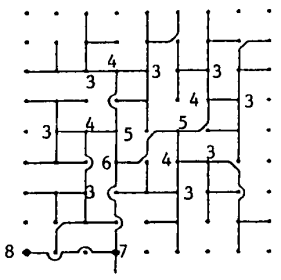


Fig. 11. A basic tile in [11].

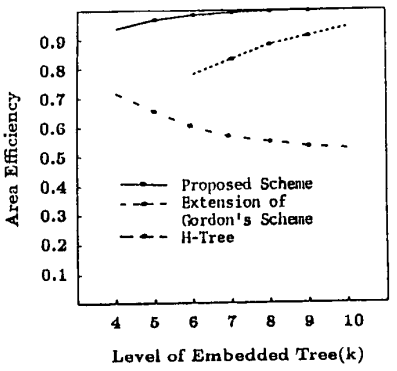


Fig. 12. Comparison of area efficiency.

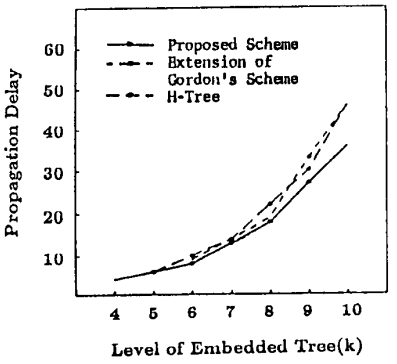


Fig. 13. Comparison of propagation delay.

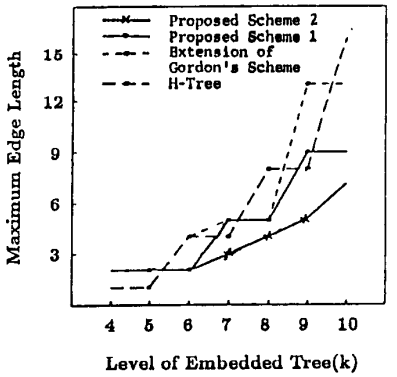


Fig. 14. Comparison of maximum edge length.

schemes. The area efficiency of the H-tree scheme is significantly poorer and only half that of the proposed scheme when the embedded tree is large. The area efficiency of a layout based on Gordon's scheme, while better than an H-tree layout, is not as good as that of the proposed scheme, particularly for moderate (practical) size trees (6–10 level). Since Gordon's design also achieves 100 percent area utilization asymptotically, this difference is smaller for larger trees.

D. Comparison of PD and MEL

Fig. 13 shows the propagation delay of the three schemes over a range of tree sizes. Recall that we take the sum of the differences of the coordinates of two processors to be the distance between them. This method of measuring distance is valid only when the connection between two communicating nodes is always the shortest possible in a channel routing scheme. For a direct extension of Gordon's scheme, this leads to a nonplanar interconnect structure which cannot be accommodated in our channel model. The planar interconnection as presented in [11] requires longer interconnections. For example, the actual distance between two nodes at level 6 and 7 in Fig. 11 from [11] is 5 for a planar design, with channel routing while our model computes this to be only 3. When we take into account this additional interconnect length, the actual propagation delay is longer than that of our design. Fig. 13 also includes a plot of propagation delay for the extended Gordon scheme.

We compare the maximum edge length in Fig. 14. As mentioned in the previous section, the basic scheme (scheme 1) proposed in Section II is optimum for these criteria for trees of up to six levels because all nodes are directly connected. We also plot the maximum edge length for the modified scheme (scheme 2) in Section IV.

E. Layouts with Wide Buses

In the entire discussion so far, we have assumed that bus widths are negligible as compared to a processor edge. This assumption will generally hold if processors are large. Notice, however, that the basic embedding scheme presented in Section II employs only a single bus in the channel between the processors. Without considering this bus area, in Fig. 12 we find that the area efficiency of our scheme is almost twice as good as that for the H-tree scheme (which, however, allows for the interconnect area by assuming unit bus and processor dimensions). Thus, for bus widths up to approximately $(1 - \sqrt{2})$ or about 40 percent of a processor edge dimension, our layouts will be more efficient than H-tree layouts while displaying much better propagation delay and maximum edge lengths. (This break-even point will be slightly lower for the modified scheme in Section IV-B because it requires two buses in a few of the interconnect channels.) If bus widths are greater than 40 percent of a processor edge, H-tree layouts will generally by more area efficient, although our designs may still display better performance.

The above comparisons suggest that the proposed tree layout scheme is generally best on all three criteria for large processors and narrow bus widths.

VI. Layouts for Defect-Tolerant Designs

Many projected applications of binary tree architecture are expected to require significantly more silicon area than can be fabricated defect-free with viable yield. As mentioned in the Introduction, for such large area and even wafer scale designs several researchers have proposed designs that include a fault tolerance capability, achieved by employing redundant processors, switches, and interconnections. It is expected that this can be used to overcome fabrication defects at manufacture, and perhaps also operational faults in the field. The technologies proposed for achieving the restructuring following manufacture and testing include electrical and laser programmed fuses, laser welding, E-beam programming of electrical switches, and discretionary wiring.

However, any fault-tolerant design developed to overcome fabrication defects must ensure that the restructured interconnection lengths following reconfiguration are quite short to minimize performance degradation. Since it is not known *a priori* as to which links will have to be restructured to bypass failed nodes, for adequate performance all interconnections must be implemented with powerful driver ciruits capable of driving the worst case restructured interconnections with acceptable delay. If restructured interconnections are long, this can impose very substantial area, power, and delay penalties on the design. Poor performance from long interconnects can negate one of the most important benefits of single silicon designs.

While a number of fault-tolerant designs for binary trees have been proposed, only the modular designs [18], [19] do not require long interconnections for restructuring. In [19], we show that the yield enhancement capabilities of modular designs can approach that of the best nonmodular design (SOFT [20] and the Cluster Proof Scheme [21]). We now show how this modular design can be efficiently laid out using the layout scheme presented in this paper.

Consider the schematic of a fault-tolerant modular tree design from [18] shown in Fig. 15(a). each module in the figure effectively implements two brother nodes in the overall tree, although for fault tolerance, it may contain any number of additional spare processors. Fig. 15(b) shows two examples of how a four-level tree is configured in the presence of faulty processors in a design with a single spare processor in each module.

Notice in Fig. 15(a) that the modules themselves (without the root node) form a binary tree. Although there are two links connecting a parent with each child in this module tree, these can be considered a single multilink interconnection. Thus, the module tree can be efficiently implemented using the layout approach presented in this paper. However, the layout must also provide space for the root node of the overall tree which is not part of the module tree. This can be done by locating the empty slot in the layout next to the root node of the module tree as in Fig. 5(b) and (c). The root node can then be located in this slot. Since only a single processing element has to be placed in this array position as compared to a complete module with three or more processors in the other positions, there may be sufficient area available to lay out the output pads along with this root node for off-chip connections.

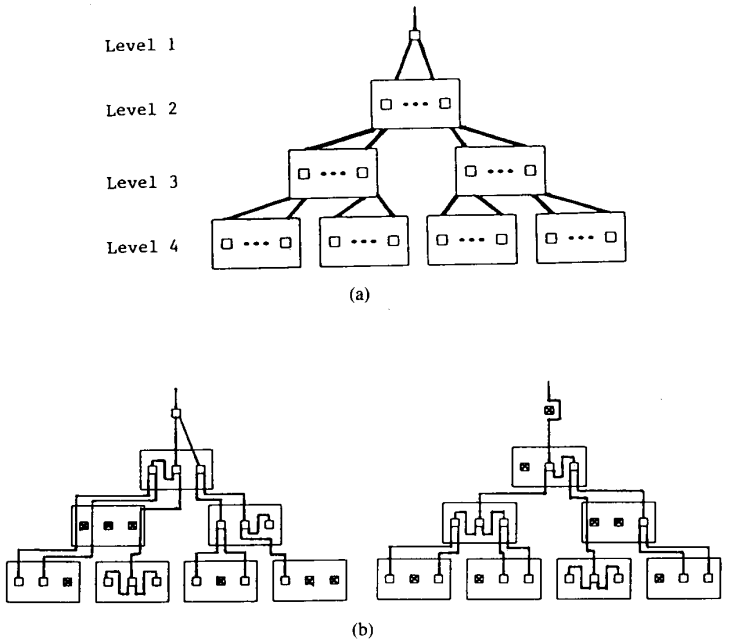


Fig. 15. Fault-tolerant modular tree design. (a) Proposed fault-tolerant modular tree architecture. (b) Examples of configured trees with 50 percent PE redundancy. (Note: Failed PE's are marked ☒.)

VII. Conclusion

We have proposed a scheme for efficiently laying out a large binary tree architecture by embedding the complete binary tree in a two-dimensional processor array under a wafer scale model [23]. Unlike many earlier layout strategies that assumed unit processors and interconnection widths, our approach is suited to large processors with relatively narrow bus widths. We show that our scheme is asymptotically optimal in area efficiency and propagation delay, and superior in these attributes to earlier designs. More importantly, the porposed scheme also displays near 100 percent area efficiency for practical size tree architectures, along with short root-to-leaf propagation delay and edge lengths. Our layouts can also incorporate defect-tolerant designs to overcome fabrication defects in large area and wafer scale implementations. Thus, using our new design, tree architectures can be layed out in VLSI more efficiently than before, with shorter internode links and hence significantly better performance.

References

[1] J. Bentley and H. T. Kung, "A tree machine for searching problems," in *Proc. Int. Conf. Parallel Processing*, Aug. 1979, pp. 257-266.

[2] M. A. Bonncelli *et al.*, "A VLSI tree machine for relational data bases," in *Proc. 10th Comput. Architecture Symp.*, June 1983, pp. 67-73.

[3] T. A. Ottman, A. L. Rosenberg, and L. J. Stockmeyer, "A dictionary machine (for VLSI)," *IEEE Trans. Comput.*, vol. C-31, pp. 892-898, Sept. 1982.

[4] A. M. Despain and D. A. Patterson, "X-tree: A tree structured multiprocessor computer architecture," in *Proc. Fifth Comput. Architecture Symp.*, Apr. 1978, pp. 144-150.

[5] R. P. Brent and H. T. Kung, "On the area of binary tree layout," *Inform. Contr.*, vol. 65, pp. 45-52, 1982.

[6] M. S. Paterson, W. L. Ruzzo, and L. Snyder, "Bounds on minimax edge length for complete binary trees," in *Proc. 13th ACM Symp. Theory Comput.*, 1981, pp. 293-299.

[7] L. G. Valiant, "Universality considerations in VLSI circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 135-140, Feb. 1981.

[8] H. Y. Youn and A. D. Singh, "On area efficient and fault tolerant tree embedding in VLSI," in *Proc. Int. Conf. Parallel Processing*, 1987, pp. 171-178.

[9] E. 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.

[10] D. Gordon *et al.*, "Embedding tree structures in VLSI hexagonal arrays," *IEEE Trans. Comput.*, vol. C-33, pp. 104-107, Jan. 1984.

[11] D. Gordon, "Efficient embeddings of binary trees in VLSI arrays," *IEEE Trans. Comput.*, vol. C-36, pp. 1009-1018, Sept. 1987.

[12] A. Mukhopadhyay and R. K. Guha, "Embedding a tree in the nearest neighbor array," in *Proc. Int. Conf. Parallel Processing*, 1981, pp. 261-263.

[13] M. S. Lee and G. Frieder, "Massively fault-tolerant cellular array," in *Proc. Int. Conf. Parallel Processing*, 1986, pp. 343-350.

[14] L. Snyder, "Introduction to the configurable, highly parallel computer," *IEEE Computer*, vol. 15, pp. 47-56, Jan. 1982.

[15] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980, ch. 8.

[16] J. F. McDonald *et al.*, "The trials of WSI," *IEEE Spectrum*, pp. 32-39, Oct. 1984.

[17] B. Chazelle and L. Monier, "A model of computation for VLSI with related complexity results," in *Proc. 13th ACM Symp. Theory Comput.*, 1981.

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

[19] ——, "A modular binary tree architecture with shared spares," *IEEE Trans. Comput.*, to be published.

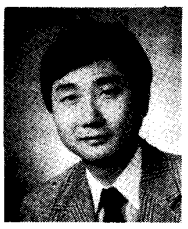
[20] M. B. Lowrie and W. K. Fuchs, "Reconfigurable tree architectures using subtree oriented fault tolerance," *IEEE Trans. Comput.*, vol. C-36, pp. 1172-1182, Oct. 1987.

[21] 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.

[22] A. L. Rosenberg, "The Diogenes approach to testable fault-tolerant arrays of processors," *IEEE Trans. Comput.*, vol. C-32, pp. 902-910, 1983.

[23] T. Leighton and C. E. Leiserson, "Wafer scale integration of systolic arrays," *IEEE Trans. Comput.*, vol. C-34, pp. 448-461, May 1985.

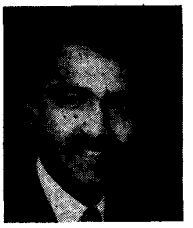
[24] F. M. Rhodes, "Performance characteristics of the RVLSI technology," in *Proc. IFIP Workshop Wafer-Scale Integration*, Genoble, 1986.
[25] W. L. Ruzzo and L. Snyder, "Minimum edge length planer embeddings of trees," in *VLSI Systems and Computation*, Kung, Sproull, and Steel, Eds. Rockville, MD: Computer Science, 1981.
[26] I. Koren, "A reconfigurable and fault-tolerant VLSI multiprocessor array," in *Proc. 8th Annu. Symp. Comput. Architecture*, May 1981, pp. 426–441.



Hee Yong Youn (S'77–M'89) received the B.S. and M.S. degrees in electrical engineering from Seoul National University, Seoul, Korea, in 1977 and 1979, respectively, and the Ph.D. degree in computer engineering from the University of Massachusetts, Amherst, in 1988.

He is currently an Assistant Professor in the Department of Computer Sciences, University of North Texas, Denton, and a Supervisor of Digital Logic Lab. From 1979 to 1984 he was on the Research Staff of the Gold Star Precision Central Research Laboratories, Korea. There he was engaged in the R&D of high-quality computer-controlled electronic systems for both military and commercial applications, and he was in charge of design and testing of the central processing unit. Among the projects he joined in that period, the computerized color radar system received the Annual Presidential Award for the Best Development in 1983. His current research interests include VLSI design, fault-tolerant computing, computer architecture for parallel processing, and testing and diagnosis. He received the Best Paper Award at the 1988 IEEE International Conference on Distributed Computing Systems.

Dr. Youn is a member of the Association for Computing Machinery and the IEEE Computer Society.



Adit D. Singh (S'81–M'83) received the B. Tech. degree from the Indian Institute of Technology, Kanpur, in 1976 and the M.S. and Ph.D. degrees from Virginia Tech, Blackburg, in 1978 and 1982, all in electrical engineering.

From 1978 to 1983, he was an Instructor in Electrical Engineering at Virginia Tech. Since 1983 he has been with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, where he is currently Associate Professor. He teaches courses in VLSI design, computer systems architecture, and fault-tolerant computing. He has also served as a consultant to AT&T Bell Laboratories, Digital Equipment Corporation, Data General, and Control Data Corporation. His research interests include high-performance VLSI and wafer scale systems, fault-tolerant computing, and multiple-valued logic design. He was co-organizer and Vice-Chair of the 1988 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems. He also received the Best Paper Award at the 1988 International Conference on Distributed Computing Systems.

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