Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping

Valavan Manoharajah, Stephen D. Brown, *Member, IEEE*, and Zvonko G. Vranesic, *Life Senior Member, IEEE*

Abstract—In this paper, an iterative technology-mapping tool called IMap is presented. It supports depth-oriented (area is a secondary objective), area-oriented (depth is a secondary objective), and duplication-free mapping modes. The edge-delay model (as opposed to the more commonly used unit-delay model) is used throughout. Two new heuristics are used to obtain area reductions over previously published methods. The first heuristic predicts the effects of various mapping decisions on the area of the final solution, and the second heuristic bounds the depth of the mapping solution at each node. In depth-oriented mode, when targeting five lookup tables (LUTs), IMap obtains depth optimal solutions that are 44.4%, 19.4%, and 5% smaller than those produced by FlowMap, CutMap, and DAOMap, respectively. Targeting the same LUT size in area-oriented mode, IMap obtains solutions that are 17.5% and 9.4% smaller than those produced by duplication-free mapping and ZMap, respectively. IMap is also shown to be highly efficient. Runtime improvements of between 2.3× and 82× are obtained over existing algorithms when targeting five LUTs. Area and runtime results comparing IMap to the other mappers when targeting four and six LUTs are also presented.

Index Terms—Circuit optimization, circuit synthesis, design automation, field programmable gate arrays, logic design.

I. INTRODUCTION

THE PROCESS of field programmable gate array (FPGA) technology mapping converts a circuit composed of simple gates into a circuit composed of lookup tables (LUTs) suitable for implementation on an FPGA. The mapping procedure attempts to reduce area, delay, or a combination of area and delay in the final LUT network. Recent work has also considered the reduction of power during the mapping process.

Most of the existing literature [1] on technology mapping uses the term depth to refer to delay, and we adopt the same terminology in describing our own work.

Technology mapping is often treated as a covering problem. For example, consider the process of technology mapping as illustrated in Fig. 1. Fig. 1(a) illustrates the initial gate level network, Fig. 1(b) illustrates a possible covering of the initial network using four-input LUTs, and Fig. 1(c) illustrates the LUT network produced by the covering. In the mapping given,
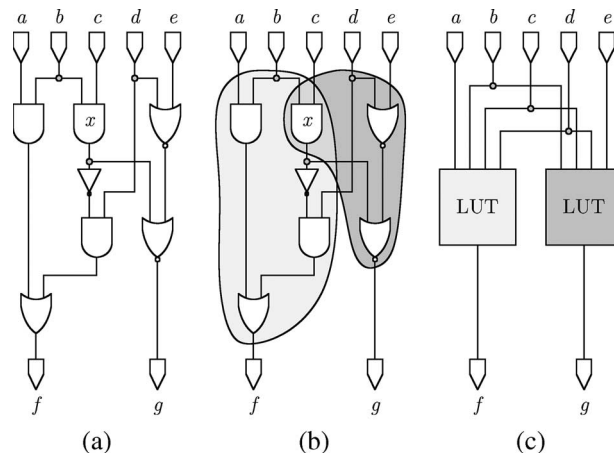
Fig. 1. Technology mapping as covering problem. (a) Initial netlist. (b) Possible covering. (c) Mapping produced by the covering.

the gate-labeled x is covered by both LUTs and is said to be duplicated. Techniques that map for depth use large amounts of duplication to obtain solutions of reduced depth, while techniques that map for area limit the use of duplication as it often increases the area of the mapped solutions.

The problem of mapping for depth can be solved optimally in polynomial time using a dynamic programming procedure [2], [4]. However, the area-minimization problem was shown to be nondeterministic polynomial-time hard (NP-hard) for LUTs of size three and greater [5]–[7]. Thus, heuristics are necessary to solve the area-minimization problem. Early work considered the decomposition of circuits into a set of trees, which were then mapped for area [8]–[10]. The area-minimization problem for trees is much simpler and can be solved optimally using dynamic programming. However, real circuits are rarely structured as trees, and tree decomposition prevents much of the optimization that can take place across tree boundaries. In a duplication-free mapping, each gate in the initial circuit is covered by a single LUT in the mapped circuit. The area-minimization problem in the duplication-free mapping can be solved optimally by decomposing the circuit into a set of maximum fan-out free cones (MFFCs), which are then mapped for area [3]. Although the area minimal duplication-free mapping is significantly smaller than the area minimal tree mapping, the controlled use of duplication can lead to further area savings. In [13], heuristics are used to mark a set of gates as duplicable. Then, area optimization is considered within an extended fan-out free cone (EFFC), where an EFFC is an MFFC that has been extended to include duplicable gates.

Area-minimization heuristics are typically used in concert with techniques that produce depth-optimal mapping solutions. In FlowMap-r [3], following a depth-optimal mapping procedure, noncritical parts of the circuit are remapped with a duplication-free mapper. In CutMap [11], two strategies are used for selecting the gates covered by an LUT. Critical parts of the circuit are mapped using a depth-minimizing strategy, and noncritical parts are mapped using a cost-minimizing strategy that encourages LUT sharing. A similar two-pronged strategy is also used in PowerMinMap [12], where power, rather than area, is minimized on the noncritical parts of the circuit.

This paper describes the IMap technology-mapping tool, which incorporates several novel features. First, it uses iteration as a method of gathering data to guide the mapping process. Second, it introduces two heuristics to solve the area-minimization problem. The first heuristic, which is called area flow, approximates the area of a mapping solution and can be optimized using a dynamic programming formulation. Area flow is similar to the area-measurement techniques presented in [13] (applicable to MFFCs) and [25] (applicable to standard cell mapping). The second heuristic determines a depth bound that must be met by each LUT used to cover the initial circuit in order to meet some depth requirement in the mapped circuit. An area-efficient depth-bounded mapping solution can be obtained by selecting LUTs that meet the depth bound requirement while minimizing area flow. Finally, in the interest of generality, IMap uses the edge-delay model of [15] rather than the more common unit-delay model. In the edge-delay model, arbitrary delay values can be assigned to each branch of a net. These delay values may reflect an estimate of placement and routing delays or, in the case of a remapping procedure such as in [16], may reflect actual delays from a placed and routed circuit.

The heuristics used by IMap have also proven to be quite flexible. Following the initial description in [26], subsequent work has employed the heuristics within an integrated mapping environment capable of targeting both standard cells and FPGAs and able to optimize for delay, area, power, and placement [27]–[29].

II. PRELIMINARIES AND PROBLEM DEFINITION

The combinational portion of a Boolean circuit can be represented as a directed acyclic graph (DAG) $G = (V(G), E(G))$. A node in the graph $v \in V(G)$ represents a logic gate, primary input or primary output, and a directed edge in the graph $e \in E(G)$ with head $u = \text{head}(e)$ and tail $v = \text{tail}(e)$, which represents a signal in the logic circuit that is an output of gate u and an input of gate v. The set of input edges for a node v $\text{iedge}(v)$ is defined to be the set of edges with v as a tail. Similarly, the set of output edges for v $\text{oedge}(v)$ is defined to be the set of edges with v as a head. A primary input (PI) node has no input edges, and a primary output (PO) node has no output edges. An internal node has both input and output edges. The set of distinct nodes that supplies input edges to v is referred to as input nodes and is denoted as $\text{inode}(v)$. Similarly, the set of distinct nodes that connects to output edges from v is referred to as output nodes and is denoted as $\text{onode}(v)$. A node v is K-feasible if $|\text{inode}(v)| \leq K$. If every node in a graph is K-feasible, then the graph is K-bounded.
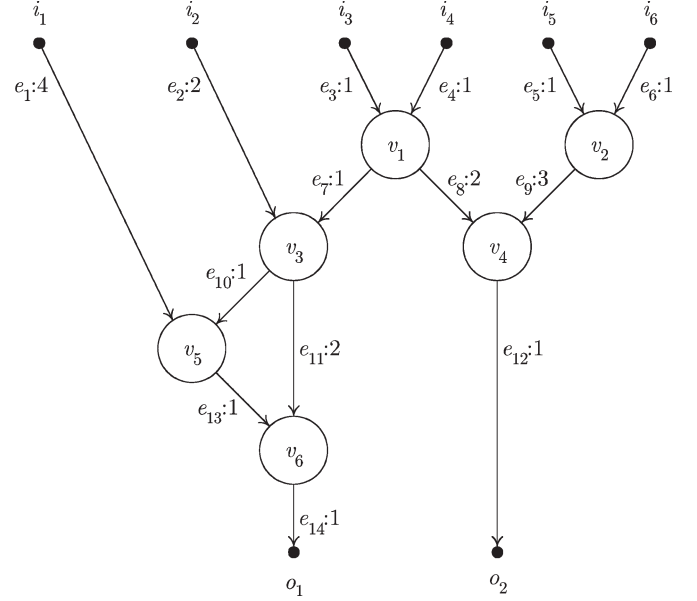

Fig. 2. Graph G.

Each edge e has an associated delay denoted $\text{delay}(e)$. The length of a path is the sum of the delays of the edges along the path. At a node v, the depth $\text{depth}(v)$ is the length of the longest path from a primary input to v, and the height $\text{height}(v)$ is the length of the longest path from a primary output to v. Both the depth for a PI node and the height for a PO node are zero. At an edge e, the depth $\text{depth}(e)$ is the length of the longest path from a primary input to e, and the height $\text{height}(e)$ is the length of the longest path from a primary output to e. Both the depth and the height of an edge include the delay due to the edge itself. The depth or height of a graph is the length of the longest path in the graph.

Fig. 2 presents a graph G that will be used as a running example to illustrate the notions defined in this section. On the graph, edges are labeled with e, PI nodes are labeled with i, PO nodes are labeled with o, and internal nodes are labeled with v. The inputs and outputs of node v_3 are

$$\text{iedge}(v_3) = \{e_2, e_7\}$$
$$\text{oedge}(v_3) = \{e_{10}, e_{11}\}$$
$$\text{inode}(v_3) = \{i_2, v_1\}$$
$$\text{onode}(v_3) = \{v_5, v_6\}.$$

Every node has two or fewer inputs, thus the graph is two bounded. The delay of each edge is specified on the graph, which is separated from the edge's label by a colon. These delays can be used to determine the depth and height of the nodes and edges in the graph as indicated in Table I. The depth (or height) of the graph is six.

Every edge and node in the graph has an associated area flow that represents an estimate of the area of the subgraph above it. Area flow is denoted as $\text{af}(\cdot)$ and is defined recursively as follows. The area flow at an edge e is given by

$$\text{af}(e) = \frac{\text{af}(\text{head}(e))}{|\text{oedge}(\text{head}(e))|} \quad (1)$$

TABLE I
DEPTH, HEIGHT, AND AREA-FLOW VALUES FOR NODES AND EDGES IN G

edge	depth	height	af	node	depth	height	af
e_1	4	6	0	i_1	0	6	0
e_2	2	5	0	i_2	0	5	0
e_3	1	5	0	i_3	0	5	0
e_4	1	5	0	i_4	0	5	0
e_5	1	5	0	i_5	0	5	0
e_6	1	5	0	i_6	0	5	0
e_7	2	4	0.5	v_1	1	4	1
e_8	3	3	0.5	v_2	1	4	1
e_9	4	4	1	v_3	2	3	1.5
e_{10}	3	3	0.75	v_4	4	1	2.5
e_{11}	4	3	0.75	v_5	4	2	1.75
e_{12}	5	1	2.5	v_6	5	1	3.5
e_{13}	5	2	1.75	o_1	6	0	3.5
e_{14}	6	1	3.5	o_2	5	0	2.5

and the area flow at a node v is given by

$$\mathrm{af}(v) = A_v + \sum_{I \in \mathrm{iedge}(v)} \mathrm{af}(i) \tag{2}$$

where the area of a node A_v is zero for primary input/output nodes and is one for internal nodes. These equations treat the area flowing into a node as the total area flowing in on the input edges. The area flowing out of a node includes the area flowing into the node as well as a component, which represents the area of the node itself. Furthermore, the area flowing out of a node is evenly divided among the outgoing edges. Column af in Table I indicates the area-flow values for the nodes and edges in G.

A cone of v C_v is a subgraph consisting of v and some of its nonPI predecessors, such that any node $u \in C_v$ has a path to v that lies entirely in C_v. Node v is referred to as the root of the cone. The size of a cone is the number of nodes and edges in the cone, and it is this parameter that determines the computational complexity of operations on the cone. At a cone C_v, the set of input edges $\mathrm{iedge}(C_v)$ is the set of edges with tail in C_v and head outside C_v, and the set of output edges $\mathrm{oedge}(C_v)$ is the set of edges with v as a head. With input and output edges so defined, a cone can be viewed as a node, and notions that were previously defined for nodes can be extended to handle cones. Notions such as $\mathrm{inode}(\cdot)$, $\mathrm{onode}(\cdot)$, $\mathrm{depth}(\cdot)$, $\mathrm{height}(\cdot)$, $\mathrm{af}(\cdot)$, and K-feasibility all have similar meanings for cones as they do for nodes. Fig. 3 highlights a set of cones in G. The largest cone is rooted at v_6 and consists of three nodes. Its inputs and outputs are

$$\mathrm{iedge}(C_{v_6}) = \{e_1, e_2, e_7\}$$
$$\mathrm{oedge}(C_{v_6}) = \{e_{14}\}$$
$$\mathrm{inode}(C_{v_6}) = \{i_1, i_2, v_1\}$$
$$\mathrm{onode}(C_{v_6}) = \{o_1\}.$$

Table II presents the depth, height, and area-flow values for the nodes and edges defined by the cones of Fig. 3. Each cone is viewed as a node, and values for depth, height, and area flow are derived accordingly.
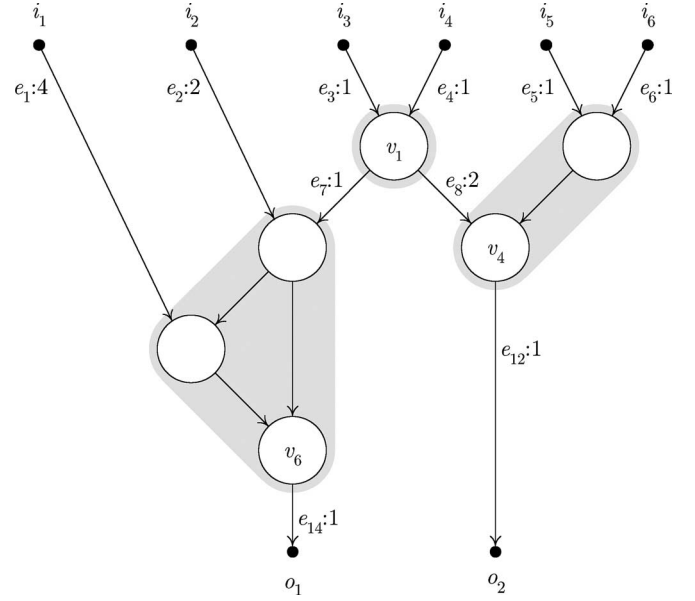

Fig. 3. Set of cones in G.

TABLE II
DEPTH, HEIGHT, AND AREA-FLOW VALUES OBTAINED
WITH SET OF CONES IN G

edge	depth	height	af	node	depth	height	af
e_1	4	5	0	i_1	0	5	0
e_2	2	3	0	i_2	0	3	0
e_3	1	4	0	i_3	0	4	0
e_4	1	4	0	i_4	0	4	0
e_5	1	2	0	i_5	0	2	0
e_6	1	2	0	i_6	0	2	0
e_7	2	2	0.5	v_1	1	3	1
e_8	3	3	0.5	v_4	3	1	1.5
e_{12}	4	1	1.5	v_6	4	1	1.5
e_{14}	5	1	1.5	o_1	5	0	1.5
				o_2	4	0	1.5

A K-input LUT (K-LUT) can implement any K-feasible cone. Thus, the technology mapping problem for LUTs is reduced to the problem of selecting a set of K-feasible cones to cover the graph in such a way that every edge is entirely within a cone or is an output edge of a cone. In the depth-oriented mapping problem, the length of the longest path through the cones selected to cover the graph is to be minimized, and in the area-oriented mapping problem, the number of cones selected to cover the graph is to be minimized. The roots of cones selected to cover the graph are said to be visible in the mapping solution generated by the covering. The cones of Fig. 3 represent a potential mapping solution for G if $K \geq 3$. Nodes v_1, v_4, and v_6 are visible in the resulting mapping solution.

For every nonroot node $v \in C_v$, if all of its output edges are also in C_v, then the cone is termed duplication free (DF-cone). A duplication-free mapping solution is one that uses DF-cones exclusively. The cones illustrated in Fig. 3 are DF-cones, and the resulting mapping solution is duplication free.

A cut $(X, \overline{X})$ is a partition of the nodes in G, such that all PI nodes are in X and all PO nodes are in $\overline{X}$. Furthermore, the cut is defined in such a way that any edge crossing the cut has a head in X and a tail in $\overline{X}$. The volume of a cut $\mathrm{vol}(X, \overline{X})$ is
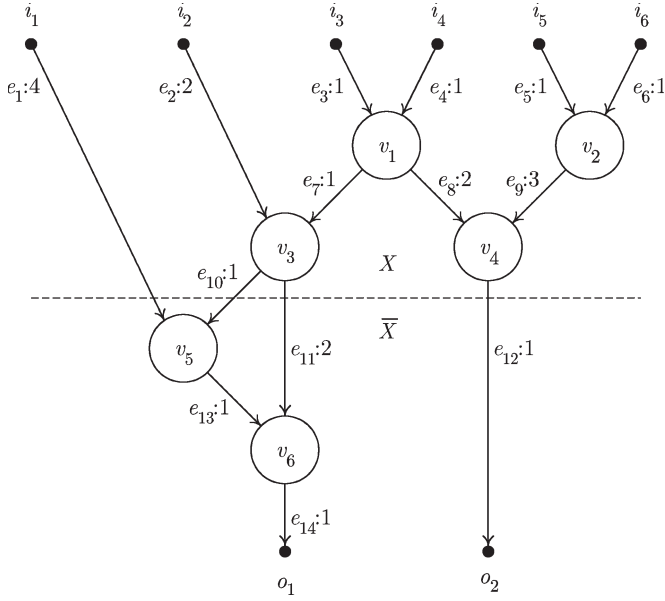
Fig. 4. Dashed line indicates cut in G, $(X, \overline{X})$.

```
1   GENERATECONES()
2   for  i ← 1 upto MaxI
3        TRAVERSEFWD()
4        TRAVERSEBWD()
5   end for
6   CONESTOLUTS()
```

Fig. 5. High-level overview of iterative technology-mapping algorithm.

the number of internal nodes in X, and the area flow of a cut $\text{af}(X, \overline{X})$ is the sum of the area flows of the edges crossing the cut. A dashed line in Fig. 4 indicates a cut in G, $(X, \overline{X})$, where X consists of the nodes v_1, v_2, v_3, and v_4, and $\overline{X}$ consists of the nodes v_5 and v_6. Edges e_1, e_{10}, e_{11}, and e_{12} cross the cut. Both the volume $\text{vol}(X, \overline{X})$ and the area flow $\text{af}(X, \overline{X})$ of the cut are four.

III. ITERATIVE TECHNOLOGY-MAPPING ALGORITHM

A. Overview

A high-level overview of IMap's iterative technology-mapping algorithm is presented in Fig. 5. First, a call to GENERATECONES generates the set of all K-feasible cones for every node in the graph. Then, a series of forward and backward graph traversals are started. The number of traversals is limited by a user-specified maximum MaxI, which was set to 20 in the experiments presented in Section V. Higher values of MaxI did not produce significantly better results. The forward traversal TRAVERSEFWD selects a cone for each node, and the backward traversal TRAVERSEBWD selects a set of cones to cover the graph. Iteration is beneficial because every backward traversal influences the behavior of the forward traversal that follows it. Finally, a call to CONESTOLUTS converts the cones selected by the final backward traversal into LUTs.

The following sections examine the steps in the algorithm in greater detail.

	M		Cone Generation	
K	M_{avg}	M_{max}	Time(s)	Ratio
4	6.5	96	6.0	0.26
5	15.6	486	19.3	0.40
6	41.5	3818	115.4	0.62

B. Generating All K-Feasible Cones

An algorithm described in [13] and [17] is used to generate all K-feasible cones in the graph. This algorithm is reviewed here. The K-feasible cones are generated as the graph is traversed in topological order from primary inputs to primary outputs. At every internal node v, new cones are generated by combining the cones at the input nodes in every possible way. Node v is added to every new cone that is generated. The new cones are then tested for K-feasibility, and any cone that is not K-feasible is discarded.

For large values of K, the generation of all K-feasible cones consumes a significant portion of the total technology-mapping time. The time consumed by cone generation is dependent on the average number of cones per node M_{avg} and the number of inputs to a cone K. A node with two inputs will generate $O(M_{\text{avg}}^2)K$-feasible cones, and any operation on a K-feasible cone (such as copying or testing for uniqueness) takes $O(K)$ time. Thus, the generation of all K-feasible cones in a graph of n nodes takes $O(KM_{\text{avg}}^2 n)$ time. Table III summarizes the values observed for M_{avg} and M_{max}, the maximum number of cones at any node for three different values of K when mapping the MCNC [20] circuits (see Section V). The table also provides the total time needed for cone generation on a 3-GHz Intel Xeon processor and a ratio indicating the time spent in cone generation as a fraction of the total time needed for technology mapping.

C. Forward Traversal

The algorithm used for forward traversal is presented in Fig. 6. During the traversal, the algorithm updates the depth and the area flow for every node and edge encountered. First, the algorithm initializes these values for PI nodes and edges attached to PI nodes. Then, the internal nodes are examined in the topological order generated by TSORT. This ordering ensures that the depth and area-flow values for a node's predecessors have been determined before the node is processed.

At each internal node v, a call to BESTCONE(v) is used to select a cone rooted at v to be used in covering v and some of its predecessors in a mapping solution. This cone then determines the depth and area flow for v and its output edges. The quality of the mapping solution is determined by the cones selected by BESTCONE. We now examine two possible ways of selecting cones.

1) Depth-Oriented Cone Selection: In the depth-oriented mapping mode, the cone with the lowest depth is selected, and if cones are equivalent in depth, then the one with the lowest area flow is selected.

```
1   for    v ∈ PI
2          depth(v) ← 0
3          af(v) ← 0
4          for    e ∈ oedges(v)
5                 depth(e) ← delay(e)
6                 af(e) ← 0
7          end for
8   end for
9   for    v ∈ TSort(V(G) − PI − PO)
10         C_v ← BestCone(v)
11         depth(v) ← depth(C_v)
12         af(v) ← af(C_v)
13         for    e ∈ oedges(v)
14                depth(e) ← delay(e) + depth(v)
15                af(e) ← af(v)/|oedges(v)|
16         end for
17  end for
```

Fig. 6. Algorithm used for forward traversal.

Fig. 7. Alternate mapping solution for Fig. 2.

Selecting the cone that minimizes the depth at each node leads to a mapping solution that is depth optimal. This is a result that is true for both the unit-delay model [2] and the edge-delay model [15].

Although this selection strategy works well during the first forward traversal, there is additional information present during the subsequent traversals that can be used to reduce the mapping area. It is assumed that the first forward traversal has established the optimal mapping depth ODepth, and a preceding backward traversal has established the height of each node. Using the optimal depth and the height of a node v, a bound can be defined on the depth of a cone C_v at v as follows:

$$depth(C_v) \leq O\text{depth} - height(v). \tag{3}$$

Cones that meet the bound requirement are considered, and among a set of cones that meet the bound requirement, cones with lower area flows are preferred. This selection strategy ensures that the mapping solutions will still achieve the optimal depth, but the greater flexibility in cone selection when the depth bound has been met leads to mapping solutions that are smaller in area.

As an example of the use of depth bounds (DB), consider two mapping iterations on the graph of Fig. 2. Assume that the first iteration has established the mapping solution of Fig. 3, and the heights have been determined as indicated in Table II. The optimal depth is five, and the height of v_4 is one; therefore, a cone at v_4 has bound $depth(C_{v_4}) \leq 4$. As long as this bound is met at v_4, a depth optimal mapping solution can be obtained. Fig. 7 indicates an alternate mapping solution, which has a depth of four at v_4 and achieves the optimal depth of five.

2) Area-Oriented Cone Selection: In the area-oriented mapping mode, the cone with the lowest area flow is selected, and if cones are equivalent in area flow, then the one with the lowest depth is selected.

Although the area flow is minimized, it is unclear how this relates to the actual area of the mapping solution. The next theorem and an observation about duplication-free mapping can be used to show that the area flow and the actual area of the mapping solution are related.
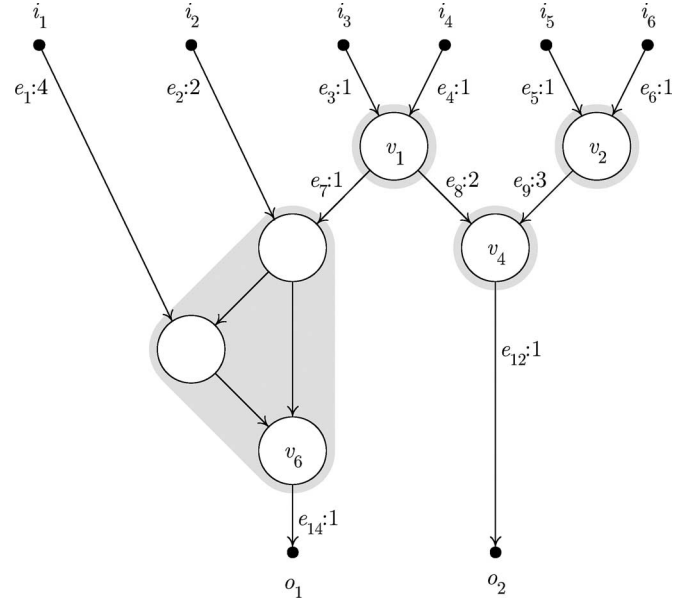
Theorem 3.1: For any cut $(X, \overline{X})$ in G, $\text{vol}(X, \overline{X}) = af(X, \overline{X})$.

Proof: The proof is by induction. In a cut of volume zero, X contains PI nodes only. All PI nodes have an area flow of zero; therefore, the cut has an area flow of zero. The relation is assumed to hold for a cut of volume n. A cut of volume $n + 1$ can be obtained from a cut of n by moving a node v from $\overline{X}$ into X. The directional constraint on the cut ensures that all input nodes of v are in X and all output nodes of v are in $\overline{X}$. Before v is moved, none of its output edges cross the cut, whereas all of its input edges cross the cut. After v is moved, all of its output edges cross the cut; whereas, none of its input edges cross the cut. By (1) and (2), the area flowing out of v on the output edges includes the area flowing in on the input edges and the area component of v, which is one. The relation holds as both the area flow and the volume are increased by one. ■

Observation 3.1: In a duplication-free mapping, a node's output edges are either covered by a single cone or not covered by any cone.

Rationale: This observation follows directly from the definition of a duplication-free mapping.

Theorem 3.1 shows that area flow can be used as a way of counting the number of internal nodes in a graph, while Observation 3.1 shows that the number of output edges for a node does not change under the duplication-free mapping. Since the number of output edges is not changed, (1) still holds and area flow can be used as a way of counting the number of duplication-free cones.

Theorem 3.1 also suggests a way of optimizing the area flow. When the graph is being traversed in topological order, the set of nodes that has been examined (X) and the set of nodes that has yet to be examined $(\overline{X})$ form a cut $(X, \overline{X})$. Every internal node that is encountered during the traversal is thought to move from $\overline{X}$ into X. If a cone with the minimum area flow is selected for each internal node as it is added to X, then the

```
1   S ← ∅
2   for   v ∈ PO
3         for   e ∈ iedges(v)
4               height(e) ← delay(e)
5         end for
6         S ← S ∪ inode(v)
7   end for
8   for   v ∈ RTSORT(V(G) − PI − PO)
9         if   v ∈ S
10              h ← max{height(e) : e ∈ oedges(v)}
11              for   u ∈ V(C_v)
12                    height(u) ← max{height(u), h}
13              end for
14              for   e ∈ iedges(C_v)
15                    height(e) ← max{height(e), delay(e) + h}
16              end for
17              S ← S ∪ inode(C_v)
18        end if
19  end for
```

Fig. 8.   Algorithm used for backward traversal.

total area flow into the PO nodes and consequently the mapping area will be minimized.

Under a general nonduplication-free mapping, the number of output edges at a node will not be known until cones have been selected for the node's successors. Thus, the area flow computed by (1) and (2) will not be equal to the mapping area. However, area flow can still be a reasonably accurate predictor of mapping area if $|\mathrm{oedge}(\cdot)|$ in (1) is replaced by an estimate $|\mathrm{oedge}(\cdot)|_{\mathrm{est}}$. During the first mapping iteration, $|\mathrm{oedge}(\cdot)|_{\mathrm{est}}$ is equal to $|\mathrm{oedge}(\cdot)|$, but during the subsequent iterations, $|\mathrm{oedge}(\cdot)|_{\mathrm{est}}$ is adjusted based on the number of output edges observed during the preceding mapping iterations. Specifically, the estimate of the number of output edges at v depends on the previous estimate $|\mathrm{oedge}(v)|'_{\mathrm{est}}$ and the number of output edges from the preceding iteration $|\mathrm{oedge}(v)|$ as follows:

$$|\mathrm{oedge}(v)|_{\mathrm{est}} = \frac{|\mathrm{oedge}(v)|'_{\mathrm{est}} + \alpha\,|\mathrm{oedge}(v)|}{1 + \alpha}. \qquad (4)$$

This equation is essentially a weighted average of two components where the contribution of the second component to the final value is determined by α. Empirically, it was observed that values of α between 1.5 and 2.5 produced the best area results. Note that there will be several nodes that are not visible in a mapping solution. These nodes will not have any output edges. The best results were obtained when the number of output edges for these nodes was assumed to be one.

D. Backward Traversal

The algorithm used for backward traversal is presented in Fig. 8. Internal nodes of the graph are visited in the reverse topological order generated by RTSORT. Although all internal nodes are visited, only the nodes that are required to be visible in a mapping solution are expanded. During the traversal, set S keeps track of the visible nodes. Initially, set S contains the nodes that connect to PO nodes as they are required to be visible. Then, at every visible node v, the cone selected for v during the preceding forward traversal C_v is used to expand S.

All input nodes of C_v are required to be visible, thus, they are added to S.

During the backward traversal, the height of all internal nodes is updated. It is assumed that before the algorithm is run, the height of all nodes and edges has been set to zero. First, the algorithm updates the heights of edges attached to PO nodes. At an internal node v, the maximum height of the node's output edges h is used to determine the height of all nodes covered by C_v. A node may be part of several cones, thus, the new height h is only assigned to a node if it is higher than its previous height. Similarly, an edge may serve as an input to several cones; thus, the height of the path through C_v is only assigned to an edge if it is higher that its previous height. The order of traversal (reverse topological order) guarantees that the edge heights have settled into their final values before they are actually used.

The height assigned to an internal node by the backward-traversal algorithm may be lower than its actual height in a mapping solution. Once again, consider the graph of Fig. 2. A potential mapping solution for the graph was given in Fig. 3. The backward-traversal algorithm determines the heights of v_6, v_5, and v_3 to be one. However, the height of one is only valid for the root of the cone v_6. A cone rooted at v_5 will have a minimum height of two, and a cone rooted at v_3 will have a minimum height of three. The heights determined by the backward traversal are only valid for those nodes that are visible in the mapping solution. The depth bound for a cone is influenced by the height assigned to its root (3), and when the depth bound is used to select cones during a depth-oriented mapping, a decision that is thought to meet the depth optimality requirement at a node may turn out to be incorrect. However, the correct decision at the nodes that were visible in the previous mapping solution ensures that the depth-optimal mapping solution is not lost.

E. Converting Cones Into LUTS

The process of converting cones into LUTs is greatly simplified by the existence of the set of visible nodes S, which is generated by the preceding backward traversal. The conversion process simply collapses the cone selected for each visible node v C_v into an LUT that implements the functionality of the cone.

F. Adding Randomness to Cone-Selection Procedure

There may be several cones, which achieve the lowest area flow value at a node. Given a particular ordering for the cones at the node and assuming that depth requirements have already been met, the cone-selection procedure simply selects the first cone that achieves the lowest area-flow value. Empirically, we have observed that the addition of a small amount of randomness to the area-flow computation leads to smaller mapping results. Specifically, area flow as defined by (2) is modified to include a small random value ϵ

$$\mathrm{af}(v) = A_v + \sum_{i \in \mathrm{iedge}(v)} \mathrm{af}(i) + \epsilon. \qquad (5)$$

TABLE IV
EFFECT OF ITERATION ON TWO VERSIONS OF IMAP: ONE THAT USES DB
ONLY AND ANOTHER THAT USES BOTH DB AND FE (DB + FE)

	DB		DB + FE	
Iteration	LUTs	af	LUTs	af
1	5090	8299.96	5090	8299.96
2	4736	5962.67	4633	4596.51
3	4716	5700.15	4579	4478.46
4	4713	5680.25	4564	4505.32
5	4713	5680.25	4554	4531.02
6	4713	5680.25	4545	4536.20
7	4713	5680.25	4544	4534.63
8	4713	5680.25	4542	4537.27

We believe that this is because the randomness forces the cone-selection procedure to consider alternative mapping solutions, which have similar area-flow costs.

IV. EFFECT OF ITERATION

Each mapping iteration in IMap gathers data used by the following iteration to determine a bound on the depth of the cone selected at each node (3) and to estimate the fanout of each node under mapping (4). Table IV presents the effect of iteration on the largest MCNC circuit `clma` when IMap is used in its depth-oriented mode. The table reports both the number of four LUTs and the total area flow seen at the primary outputs (af) for the mapping solutions produced by each iteration. For this experiment, the area-flow randomization feature described in Section III-F was turned off (we examine the effect of randomization on area in Section V-A). Data for two versions of IMap are reported. The first uses only DB and the second uses both DB and fan-out estimation (FE) (DB + FE). Both versions of IMap produce identical mapping results in the first iteration. DB and fan-out estimates are not available during the first iteration, thus, each node is mapped for minimum depth, and area flow is computed assuming that a node's fanout under mapping is identical to its unmapped fanout. When DB become available in the second iteration, significant area reductions are made by both versions. However, the first version does not make much progress in reducing area beyond the second iteration. Area-oriented cone selection is guided by area-flow values that are estimates of the actual mapping area. However, the first version does not use fan-out estimates, and the area-flow values it computes are never close to the actual mapping area. These inaccurate area-flow values hinder its progress beyond the second iteration. The second version, which uses fan-out estimates, makes a steady progress beyond the second iteration guided by increasingly accurate area-flow values.

V. RESULTS

The MCNC circuits were used to study the performance of IMap's depth-oriented and area-oriented mapping modes. Each circuit was first synthesized (SIS's [21] `script.rugged`) and decomposed into a network of two-input gates (SIS's `speed_up command`) before being technology mapped into four LUTs ($K = 4$), five LUTs ($K = 5$), and six LUTs ($K = 6$). Although four LUTs have the greatest area efficiency [18] and are the most common type of LUT present in modern

TABLE V
COMPARING IMAP TO FLOWMAP, CUTMAP, AND DAOMAP WHEN
PERFORMING DEPTH-ORIENTED MAPPING ($K = 4$)

Circuit	Depth	Area (LUTs)			
		FlowMap	CutMap	DAOMap	IMap
C6288	26	746	547	594	558
alu4	8	1376	1252	1094	1027
apex2	9	1621	1437	1262	1184
apex4	7	1190	1129	1086	1076
bigkey	4	1709	1594	1594	1594
clma	16	5900	5361	4851	4498
des	7	1672	1426	1218	1218
diffeq	13	1523	1027	852	838
dsip	5	2049	1599	1154	1154
elliptic	17	3738	2754	2105	2108
ex1010	9	2823	2698	2533	2452
ex5p	8	1158	1080	956	919
frisc	22	3782	2963	2308	2273
i10	15	1071	905	807	779
misex3	8	1441	1303	1133	1078
pdc	11	2477	2200	1916	1872
s38417	11	4387	3805	3645	3654
s38584.1	11	5395	4514	3890	3700
seq	8	1513	1341	1142	1079
spla	9	1815	1618	1400	1323
Total		79122	67986	59874	58104
Ratio		1.362	1.170	1.030	1.000
Time(s)		105.4	971.4	65.0	22.4
Ratio		4.71	43.37	2.90	1.00

FPGAs, the five-LUT mapping problem is important because some architectures allow two four LUTs to be combined into a single five LUT [19]. Furthermore, a recent FPGA from Altera [22] contains adaptive logic modules, which can be configured to implement two four LUTs, two five LUTs, or a single six LUT.

A. Depth-Oriented Mapping

We compare the area of IMap's depth-oriented mapping solutions to those produced by three other mappers, FlowMap [2], CutMap [11], and DAOMap [14]. All three mappers produce depth-optimal mapping solutions, and two of the mappers, CutMap and DAOMap, contain several heuristics that reduce the area of the depth-optimal mapping solutions. While IMap uses the edge-delay model, FlowMap, CutMap, and DAOMap use the unit-delay model. To produce comparable results, IMap assumes that every edge in the input graph is of unit delay.

Tables V–VII present the number of LUTs produced by each mapper when $K = 4$, $K = 5$, and $K = 6$, respectively. Although the tables highlight the results obtained for the 20 largest MCNC circuits, the total area and mapping time (on a 3-GHz Intel Xeon Processor) at the bottom of the table are for the entire set of MCNC circuits (202 circuits). When $K = 6$, DAOMap crashes while mapping circuit `i10`. Thus, the totals at the bottom of Table VII do not include the area for `i10` or the time needed to map it. The optimal depth for each circuit, which is achieved by all three mappers, is indicated in the second column (depth) of each table.

FlowMap produced solutions that are 36.2% ($K = 4$), 44.4% ($K = 5$), and 42.8% ($K = 6$) larger than those produced by

TABLE VI
COMPARING IMAP TO FLOWMAP, CUTMAP, AND DAOMAP WHEN
PERFORMING DEPTH-ORIENTED MAPPING ($K = 5$)

Circuit	Depth	Area (LUTs)			
		FlowMap	CutMap	DAOMap	IMap
C6288	23	839	655	634	675
alu4	7	1208	1054	916	849
apex2	8	1413	1232	1079	981
apex4	7	1204	1106	937	866
bigkey	4	1704	1362	1253	1253
clma	13	5124	4588	4152	3640
des	6	1505	1056	993	996
diffeq	10	1312	845	783	768
dsip	4	1146	921	920	923
elliptic	15	3073	2525	1936	1894
ex1010	9	2634	2309	2071	1842
ex5p	6	966	896	854	827
frisc	17	3698	2571	2026	1945
i10	12	973	847	742	692
misex3	7	1236	1096	964	885
pdc	10	2230	1970	1562	1464
s38417	9	4020	3308	3084	3039
s38584.1	9	4567	3329	2859	2750
seq	6	1299	1139	1007	957
spla	8	1516	1311	1148	1067
Total		68375	56535	49725	47357
Ratio		1.444	1.194	1.050	1.000
Time(s)		111.2	3934.0	110.5	48.0
Ratio		2.32	82.0	2.30	1.00

TABLE VII
COMPARING IMAP TO FLOWMAP, CUTMAP, AND DAOMAP WHEN
PERFORMING DEPTH-ORIENTED MAPPING ($K = 6$)

Circuit	Depth	Area (LUTs)			
		FlowMap	CutMap	DAOMap	IMap
C6288	17	539	534	609	518
alu4	6	1085	976	798	720
apex2	7	1239	1093	973	850
apex4	6	1011	935	824	769
bigkey	4	915	824	585	978
clma	11	4433	3993	3433	3108
des	4	644	555	498	501
diffeq	9	940	730	655	665
dsip	4	1137	691	690	692
elliptic	11	2492	2043	1856	1853
ex1010	8	2093	1931	1749	1577
ex5p	6	825	759	698	633
frisc	15	3115	2399	1911	1803
i10	11	858	733	—	593
misex3	6	1137	991	877	781
pdc	9	2346	1970	1318	1238
s38417	7	2777	2454	2325	2298
s38584.1	8	3986	2723	2362	2274
seq	6	1145	987	842	775
spla	7	1352	1208	1029	944
Total		56807	47746	41003	39184
Ratio		1.428	1.200	1.046	1.000
Time(s)		118.9	2274.8	224.6	158.6
Ratio		0.75	14.34	1.42	1.00

TABLE VIII
EFFECT OF VARIOUS HEURISTICS ON THE SIZE
OF THE MAPPED CIRCUIT ($K = 5$)

Heuristic	Gain (%)
Single Iteration + AF	+12.5%
+ Multiple Iterations + DB	−7.4%
+ FE	−4.6%
+ Randomization	−0.5%

TABLE IX
EFFECT OF POSTPROCESSING OPERATIONS ON AREA

K	Algorithm	w/o pp. (LUTs)	w/ pp. (LUTs)	Change (%)	Ratio
4	FlowMap	79122	71005	10.2	1.229
	CutMap	67986	66255	2.5	1.147
	DAOMap	59874	59314	0.9	1.027
	IMap	58104	57760	0.6	1.000
5	FlowMap	68375	60078	12.1	1.278
	CutMap	56535	54837	3.0	1.166
	DAOMap	49725	49384	0.7	1.050
	IMap	47357	47015	0.7	1.000
6	FlowMap	55949	48496	13.3	1.250
	CutMap	47013	45296	3.7	1.167
	DAOMap	41003	40643	0.9	1.047
	IMap	39184	38806	1.0	1.000

to map all MCNC circuits is similar regardless of LUT size. For $K = 6$, FlowMap is the fastest mapping algorithm and is approximately 25% faster than IMap. For the other values of K, IMap is significantly faster than the other three mappers with the second fastest mapper, DAOMap, being 2.3 times slower.

In addition to being the slowest mapper in the experiments, CutMap produces solutions that are 17% ($K = 4$), 19.4% ($K = 5$), and 20% ($K = 6$) larger than those produced by IMap.

DAOMap, the newest of the three mappers we compared, produces solutions that are 3%, 5%, and 4.6% larger than those produced by IMap. The results produced by these two mappers are close because they share several characteristics. First, they both generate all cones at each node and select the best cone according to a cost function. Second, they are both iterative algorithms. Finally, they both use similar methods of bounding the depth at each node.

Although most of the mapping solutions found by IMap are smaller than those found by the other three mappers, IMap finds a solution for `bigkey` that is significantly larger (978 LUTs) than the one found by DAOMap (585 LUTs) when $K = 6$. A large number of nodes need to be duplicated to find the best solution for `bigkey`, but IMap's area-flow heuristic, which encourages the sharing of nodes with several output edges, prevents many of the duplications from taking place.

Table VIII presents the effect that each heuristic in IMap has in reducing the area of the resulting circuits. The second row indicates that a version of IMap that is limited to a single iteration and uses area flow to estimate mapping area produces mapped circuits that are 12.5% larger than the best version of IMap. The remaining rows present the effect of multiple iterations, DB, FE, and randomization on the resulting area.

The rapid system prototyping (RASP) package [23] contains two postprocessing operations called mppack [24] and

IMap. FlowMap's primary objective is depth optimization, and very little is done to reduce the area. Although its area results are not as good as those produced by the other mapping algorithms, it uses a highly efficient procedure for finding cones that minimize mapping depth. For FlowMap, the time needed

TABLE X
COMPARING OPTIMAL AREA-FLOW MAPPING SOLUTION TO AREA-OPTIMAL DUPLICATION-FREE MAPPING

Circuit	$K = 4$ (LUTs)			$K = 5$ (LUTs)			$K = 6$ (LUTs)		
	DFree	ZMap	AFlow	DFree	ZMap	AFlow	DFree	ZMap	AFlow
C6288	1413	566	974	1384	644	982	1369	582	917
alu4	1053	1140	1006	896	954	824	789	828	688
apex2	1228	1317	1143	1062	1114	946	954	981	806
apex4	1092	1127	995	991	995	837	938	868	738
bigkey	1275	1156	1040	1047	699	697	1046	691	583
clma	4562	5069	4262	3898	4144	3405	3550	3476	2856
des	1236	1195	1176	1084	974	926	930	580	481
diffeq	911	946	832	833	756	746	780	701	646
dsip	1164	1377	1154	931	918	916	712	690	692
elliptic	2138	2368	2085	2015	1952	1913	1902	1872	1801
ex1010	2296	2343	2076	2091	2042	1769	1964	1755	1519
ex5p	994	1002	892	903	855	742	857	716	622
frisc	2378	2658	2239	2124	2274	1932	2000	1998	1775
i10	857	795	754	749	689	638	686	625	561
misex3	1139	1199	1065	986	1002	872	897	881	746
pdc	1852	1944	1739	1580	1617	1412	1423	1389	1195
s38417	4097	3647	3665	3595	3112	3063	3180	2314	2309
s38584.1	4083	3929	3700	3403	2973	2746	3016	2499	2246
seq	1156	1191	1073	1001	1003	879	904	893	760
spla	1361	1413	1271	1191	1206	1045	1096	1070	914
Total	61265	61097	55834	53427	49702	45452	48108	41804	37770
Ratio	1.097	1.094	1.000	1.175	1.094	1.000	1.274	1.107	1.000
Time(s)	22.4	87.5	22.4	48.0	177.3	48.0	186.4	910.3	186.4
Ratio	1.00	3.91	1.00	1.00	3.69	1.00	1.00	4.88	1.00

flowpack [2], which can be used after technology mapping to reduce area even further. Table IX presents the effect that these postprocessing operations have on mapping area. Area results obtained before (w/o pp.) and after (w/ pp.) the application of the postprocessing operations are presented. The reduction in area (change) as a result of the postprocessing operations is greatest for those algorithms that are further away from the results obtained by IMap. Both DAOMap and IMap obtain area reductions of between 0.6% and 1.0% as a result of the postprocessing operations.

B. Area-Oriented Mapping

Although the problem of finding an area-optimal mapping solution is NP-hard, an area-optimal duplication-free mapping solution can be found in polynomial time. Thus, duplication-free mapping is often used as a heuristic for minimizing mapping area [3], [13].

When IMap is used in its area-oriented mapping mode, it finds a mapping solution that minimizes area flow. Table X compares the optimal area-flow mapping solution (AFlow) to the area-optimal duplication-free mapping solution (DFree). The duplication-free mapping solutions were also produced by IMap. When IMap operates in its area-oriented mode and uses duplication-free cones exclusively, it produces an area-optimal duplication-free mapping solution. The area-flow mapping solution is also compared to the solutions produced by ZMap [23], an area-oriented technology mapper found in the RASP package. Once again, the table highlights the number of LUTs for the 20 largest MCNC circuits, but the totals at the bottom are for all MCNC circuits. The duplication-free mapping solutions are 9.7%, 17.5%, and 27.4% larger than the area-flow mapping solutions for $K = 4$, $K = 5$, and $K = 6$,

respectively. Additionally, in none of the 202 MCNC circuits was the area-flow mapping solution larger than the duplication-free mapping solution. Solutions produced by ZMap are 9.4% larger for $K = 4$ and $K = 5$ and 10.7% larger for $K = 6$. Depending on the value chosen for K, ZMap's runtimes are between 3.69 times and 4.88 times slower than those of IMap.

VI. SUMMARY

We presented an iterative technology-mapping tool called IMap. Iteration was used in conjunction with two heuristics to produce area efficient mapping solutions. The first heuristic, which is called area flow, is an estimation of the actual mapping area and can be optimized using a dynamic programming formulation. The second heuristic is a method of bounding the depth of cones selected at each node; any extra flexibility specified by the bound can be used in selecting cones that reduce mapping area.

When mapping for depth, IMap produced solutions that were between 3% and 44.4% smaller than solutions produced by FlowMap, CutMap, and DAOMap. When mapping for area, IMap produced solutions that were between 9.4% and 27.4% smaller than the optimal duplication-free mapping solutions and the solutions produced by ZMap. Runtime improvements of between 2.3× and 82× over existing algorithms were also demonstrated for $K = 4$ and $K = 5$.

REFERENCES

[1] J. Cong and Y. Ding, "Combinational logic synthesis for LUT based field programmable gate arrays," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 1, no. 2, pp. 145–204, Apr. 1996.
[2] ——, "An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 1, pp. 1–13, Jan. 1994.

[3] ——, "On area/depth tradeoff in LUT-based FPGA technology mapping," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 2, pp. 137–148, Jun. 1994.

[4] Y. Kukimoto, R. K. Brayton, and P. Sawkar, "Delay-optimal technology mapping by DAG covering," in *Proc. Des. Autom. Conf.*, San Francisco, CA, Jun. 1998, pp. 348–351.

[5] I. Levin and R. Y. Pinter, "Realizing expression graphs using table-lookup FPGAs," in *Proc. Eur. Des. Autom. Conf.*, Hamburg, Germany, Sep. 1993, pp. 306–311.

[6] A. Farrahi and M. Sarrafzadeh, "Complexity of the lookup-table minimization problem for FPGA technology mapping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 11, pp. 1319–1332, Nov. 1994.

[7] S. Zhang, D. M. Miller, and J. M. Muzio, "Notes on the complexity of the lookup-table minimization problem for FPGA technology mapping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 12, pp. 1588–1590, Dec. 1996.

[8] K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching," in *Proc. Des. Autom. Conf.*, Miami Beach, FL, 1987, pp. 341–347.

[9] R. J. Francis, J. Rose, and Z. G. Vranesic, "Technology mapping of lookup table-based FPGAs for performance," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, Santa Clara, CA, Nov. 1991, pp. 568–571.

[10] ——, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs," in *Proc. ACM/IEEE Des. Autom. Conf.*, San Francisco, CA, Jun. 1991, pp. 227–233.

[11] J. Cong and Y.-Y. Hwang, "Simultaneous depth and area minimization in LUT-based FGPA mapping," in *Proc. ACM Int. Symp. FPGAs*, Monterey, CA, Feb. 1995, pp. 68–74.

[12] H. Li, S. Katkoori, and W. K. Mak, "Power minimization algorithms for LUT based FPGA technology mapping," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 1, pp. 33–51, Jan. 2004.

[13] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proc. ACM Int. Symp. FPGAs*, Monterey, CA, Feb. 1999, pp. 29–35.

[14] D. Chen and J. Cong, "DAOMap: A depth-optimal area optimization mapping algorithm for FPGA designs," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 2004, pp. 752–759.

[15] H. Yang and D. F. Wong, "Edge-Map: Optimal performance driven technology mapping for iterative LUT based FPGA designs," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 1994, pp. 150–155.

[16] J. Y. Lin, A. Jagannathan, and J. Cong, "Placement-driven technology mapping for LUT-based FPGAs," in *Proc. ACM Int. Symp. FPGAs*, Monterey, CA, Feb. 2003, pp. 121–126.

[17] M. Schlag, J. Kong, and P. K. Chan, "Routability-driven technology mapping for lookup table-based FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 1, pp. 13–26, Jan. 1994.

[18] J. Rose, R. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE J. Solid-State Circuits*, vol. 25, no. 5, pp. 1217–1225, Oct. 1990.

[19] *Virtex II Platform FPGA Data Sheet*, 2002, San Jose, CA: Xilinx Inc. [Online]. Available: http://www.xilinx.com/apps/virtexapp.htm

[20] Collaborative Benchmarking Laboratory, *LGSynth93 Benchmark Suite*. [Online]. Available: http://www.cbl.ncsu.edu/www/

[21] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Univ. California at Berkeley, Berkeley. Tech. Report, Memo. No. UCB/ERL M92/41, 1992.

[22] Altera Inc., *Stratix II Device Handbook (Complete Two-Volume Set)*. Version 2.0, Jan. 2005.

[23] J. Cong, J. Peck, and Y. Ding, "RASP: A general logic synthesis system for SRAM-based FPGAs," in *Proc. ACM Int. Symp. FPGAs*, Monterey, CA, Feb. 1996, pp. 137–143.

[24] K. C. Chen, J. Cong, Y. Ding, A. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA technology mapping for delay optimization," *IEEE Des. Test Comput.*, vol. 9, no. 3, pp. 7–20, Sep. 1992.

[25] A. Lu, G. Stenz, and F. M. Johannes, "Technology mapping for minimizing gate and routing area," in *Proc. Conf. Des., Autom. Test Eur.*. Paris, France: Le Palais des Congres de Paris, 1998, pp. 664–669.

[26] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," in *Proc. Int. Workshop Logic and Synthesis*, Temecula Creek, CA, 2004, pp. 14–21.

[27] MVSIS Group, *MVSIS: Multi-Valued Logic Synthesis System*, Berkeley: Univ. California Berkeley. [Online]. Available: http://www-cad.eecs.berkeley.edu/mvsis/

[28] A. Mishchenko, S. Chatterjee, and R. Brayton, "An integrated technology mapping environment," in *Proc. Int. Workshop Logic and Synthesis*, Lake Arrowhead, CA, 2005, pp. 383–390.

[29] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," in *Proc. Int. Workshop Logic and Synthesis*, Lake Arrowhead, CA, 2005, pp. 375–382.

Valavan Manohararajah received the B.A.Sc., M.A.Sc., and Ph.D. degrees in computer engineering from University of Toronto, Toronto, ON, Canada, in 1998, 2001, and 2005, respectively.

He is currently a member of technical staff with Altera Toronto Technology Center. His current research interests include synthesis, placement, and verification of digital circuits.

Stephen D. Brown (M'71) received the B.A.Sc. degree from the University of New Brunswick, Fredericton, NB, Canada, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada, all in electrical engineering.

He is a Tenured Professor with the University of Toronto, a Director of research and development with the Altera Toronto Technology Center, Toronto, and the Director of the worldwide University Program with Altera. He has authored more than 60 scientific publications and coauthored three text books entitled *Fundamentals of Digital Logic with VHDL Design* 2nd ed. (McGraw-Hill, 2004), *Fundamentals of Digital Logic with Verilog Design* 1st ed. (McGraw-Hill, 2002), and *Field-Programmable Gate Arrays* (Kluwer Academic, 1992). His research interests include computer-aided design algorithms, field-programmable very-large-scale-integration technology, and computer architecture.

Mr. Brown was a recipient of a number of awards for excellence in teaching. He was a recipient of the Canadian Natural Sciences and Engineering Research Council's 1992 Doctoral Prize for the Best Doctoral Thesis in Canada.

Zvonko G. Vranesic (S'67–M'68–SM'84–LS'04) received the B.A.Sc., M.A.Sc., and Ph.D. degrees, all in electrical engineering, from University of Toronto, Toronto, ON, Canada.

From 1963 to 1965, he worked as a Design Engineer with the Northern Electric Co., Ltd., Bramalea, ON. In 1968, he joined with University of Toronto, where he is currently a Professor Emeritus in the Department of Electrical and Computer Engineering. During the 1978–1979 academic year, he was a Senior Visitor with the University of Cambridge, England, and during 1984–1985, he was with University of Paris. From 1995 to 2000, he served as a Chair with the Division of Engineering Science, University of Toronto. He is also involved in research and development at the Altera Toronto Technology Center. His current research interests include computer architecture, field-programmable VLSI technology, and multiple-valued logic systems. He has coauthored five books: *Computer Organization*, 5th ed. (McGraw-Hill, 2001), *Fundamentals of Digital Logic with VHDL Design*, 2nd ed. (McGraw-Hill, 2004); *Fundamentals of Digital Logic with Verilog Design*, 1st ed. (McGraw-Hill, 2002); *Microcomputer Structures*, (Oxford Univ. Press, 1989) and *Field-Programmable Gate Arrays* (Kluwer Academic, 1992).

Dr. Vranesic was the Chairman of the 3rd International Symposium on Multiple-Valued Logic in 1973 and of the 18th International Symposium on Computer Architecture in 1991. In 1990, he received the Wighton Fellowship for "innovative and distinctive contributions to undergraduate laboratory instruction."