Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping

Valavan Manohararajah, Stephen D. Brown and Zvonko G. Vranesic
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, CANADA
manohv|brown|zvonko@eecg.toronto.edu

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 common 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 5-LUTs, IMap obtains depth optimal solutions that are 13.3% and 12.5% smaller than those produced by CutMAP and FlowMAP-r0, respectively. Targetting the same LUT size in area-oriented mode, IMap obtains solutions that are 13.7% smaller than those produced by duplication-free mapping.

1. INTRODUCTION

Previous work has shown that the depth minimization problem in lookup table (LUT) based FPGA technology mapping can be solved optimally in polynomial time using a dynamic programming procedure [1, 3]. However, the area minimization problem was shown to be NP-hard for LUTs of size four and greater [4, 5]. 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 [6, 7, 8]. 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 duplication-free mapping can be solved optimally by decomposing the circuit into a set of maximum fanout free cones (MFFCs) which are then mapped for area [2]. 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 [10], heuristics are used to mark a set of gates as *duplicable*. Then area optimization is considered within an extended fanout 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 [2], following a depth-optimal mapping procedure, noncritical parts of the circuit are remapped with a duplication-free mapper. In CutMAP [9], two strategies are used for selecting the gates covered by a 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.

This work 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, called *area flow*, is closely related to 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 [10] (applicable to MFFCs) and [20] (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 [11] 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 [12], may reflect actual delays from a placed and routed circuit.

2. 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, $u = head(e)$, and tail, $v = tail(e)$, 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, $iedge(v)$, is defined to be the set of edges with v as a tail. Similarly, the set of *output edges* for v, $oedge(v)$, 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 edges and output edges. The set of distinct nodes that supply input edges to v are referred to as *input nodes* and is denoted $inode(v)$. Similarly, the set of distinct nodes that connect to output edges from v are referred to as *output nodes* and is denoted $onode(v)$. A node v is *K-feasible* if $|inode(v)| \leq K$. If every node in a graph is K-feasible then the graph is *K-bounded*.

Each edge e has an associated delay denoted $delay(e)$. The length of a path is the sum of the delays of the edges along the path. At a node v, the depth, $depth(v)$, is the length of the longest path from a primary input to v and the height, $height(v)$, 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, $depth(e)$, is the length of the longest path from a primary input to e and the height, $height(e)$, 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.

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

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

and the area flow at a node v is given by

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

where the area of a node, A_v, is zero for primary input/output nodes and is 1 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.

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, $iedge(C_v)$, are the set of edges with a tail in C_v and the set of output edges, $oedge(C_v)$, are the set of edges with v as a head. With input edges 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 $inode(\cdot)$, $onode(\cdot)$, $depth(\cdot)$, $height(\cdot)$, $af(\cdot)$, and K-feasibility all have similar meanings for cones as they do for nodes.

A K-input lookup table (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.

For every non-root 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.

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, $vol(X, \overline{X})$ is the number of internal nodes in X and the area flow of a cut, $af(X, \overline{X})$, is the sum of the area flows of the edges crossing the cut.

A few examples will clarify the notions defined above. Figure 1 presents a graph, G, where 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

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

Every node has two or fewer inputs, thus the graph is 2-bounded. The delay of each edge is specified on the graph, 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 1. The depth (or height) of the graph is 6. Table 1 also provides the area flow values for the nodes and edges in the graph. A dashed line in Figure 1 indicates a cut, $(X, \overline{X})$, 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, $vol(X, \overline{X})$, and the area flow, $af(X, \overline{X})$, of the cut are 4. Figure 2 highlights three DF-cones in G. These cones represent a potential mapping solution for G if $K \geq 3$. Nodes v_1, v_4 and v_6 are visible in the mapping solution generated by the cones. The inputs and outputs of the cone rooted at v_6 are

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

Table 2 presents the depth, height, and area flow values for

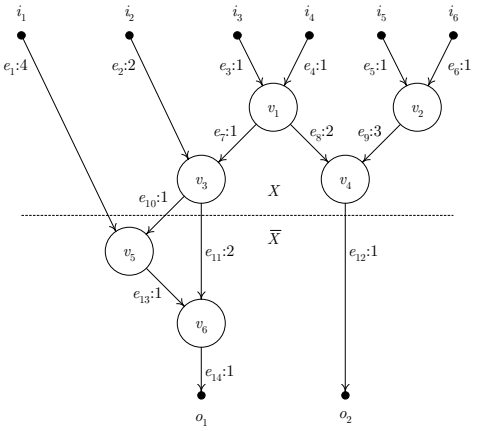

Figure 1: A graph G with a cut, $(X, \overline{X})$, indicated by the dashed line.

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

Table 1: Depth, height, and area flow values for nodes and edges in G.

the nodes and edges defined by the cones of Figure 2. Each cone is viewed as a node and values for depth, height, and area flow are derived accordingly.

3. AN ITERATIVE TECHNOLOGY MAPPING ALGORITHM

3.1 Overview

A high level overview of IMap's iterative technology mapping algorithm is presented in Figure 3. 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 8 in the experiments presented in Section 5. 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 back-

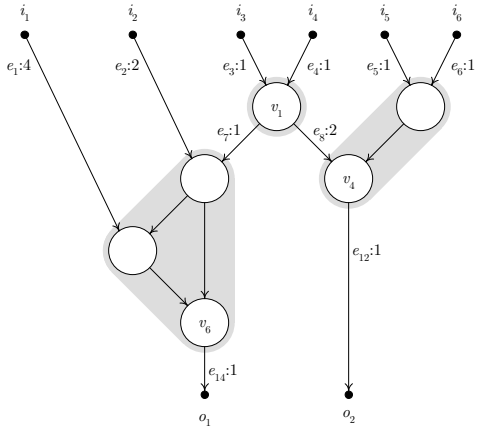

Figure 2: A set of cones in G.

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

Table 2: Depth, height, and area flow values obtained with a set of cones in G.

```
1  GenerateCones()
2  for   i ← 1 upto MaxI
3        TraverseFwd()
4        TraverseBwd()
5  end for
6  ConesToLUTs()
```

Figure 3: A high level overview of the iterative technology mapping algorithm.

K	m	c
4	6.47	4.63
5	15.1	7.37

Table 3: Values of m and c for $K = 4$ and $K = 5$.

ward traversal into LUTs.

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

3.2 Generating all K-Feasible Cones

An algorithm described in [13, 10] 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.

The generation of all K-feasible cones consumes most of the execution time in IMap. When mapping the MCNC circuits (see Section 5), cone generation consumes 80% and 98% of the overall execution time when $K = 4$ and $K = 5$, respectively. The time consumed by cone generation is dependent on the average number of cones per node, m, and the average size of a cone, c. The generation of new cones at a node with two inputs takes $O(cm^2)$ time and the test for K-feasibility takes $O(c)$ time. Thus, the generation of all K-feasible cones in a graph of n nodes takes $O(cm^2n)$ time. Table 3 indicates the values of m and c observed on the MCNC circuits when mapping with $K = 4$ and $K = 5$.

3.3 Forward Traversal

The algorithm used for forward traversal is presented in Figure 4. 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. 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

```
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
```

Figure 4: The algorithm used for forward traversal.

ways of selecting cones.

3.3.1 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.

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 [1] and the edge delay model [11].

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

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

Cones that meet the bound requirement are preferred 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.

3.3.2 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 two theorems can be used to show that the that the area flow and the actual area of the mapping solution are related.

THEOREM 1. *For any cut, $(X, \overline{X})$, in G, $vol(X, \overline{X}) = af(X, \overline{X})$.*

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 Equations 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. □

THEOREM 2. *In a duplication-free mapping, a node's output edges are either covered by a single cone or not covered by any cone.*

PROOF. Follows directly from the definition of a duplication-free mapping. □

Theorem 1 shows that area flow can be used as a way of counting the number internal nodes in a graph while Theorem 2 shows that the number of output edges for a node does not change under duplication-free mapping. Since the number of output edges is not changed, Equation 1 still holds and area flow can be used as a way of counting the number of duplication free cones.

Theorem 1 also suggests a way of optimizing the area flow. When the graph is being traversed in topological order, the set of nodes that have been examined (X) and the set of nodes that have 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 total area flow into the PO nodes and consequently the mapping area will be minimized.

Under a general non-duplication-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 Equations 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 $|oedge(\cdot)|$ in Equation 1 is replaced by an estimate, $|oedge(\cdot)|_{est}$. During the first mapping iteration, $|oedge(\cdot)|_{est}$ is equal to $|oedge(\cdot)|$ but during the subsequent iterations, $|oedge(\cdot)|_{est}$ 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, $|oedge(v)|'_{est}$, and the number of output edges from the preceding iteration, $|oedge(v)|$, as follows

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

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. Values of α between 1.5 and 2.5 were found to work well when the number of iterations was limited to 8. 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.

3.4 Backward Traversal

The algorithm used for backward traversal is presented in Figure 5. 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 Figure 1. A potential mapping solution for the graph was given in Figure 2. The backward traversal algorithm determines the heights of v_6, v_5 and v_3 to be 1. However, the height of 1 is only valid for the root of the cone, v_6. A cone rooted at v_5 will have a minimum height of 2 and a cone rooted at v_3 will have a minimum height of 3. 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 (Equation 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.

3.5 Converting Cones into LUTs

The process of converting cones into LUTs is greatly simplified by the existence of the set of visible nodes, S, generated by the preceding backward traversal. The conversion process simply collapses the cone selected for each visible node

```
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
```

Figure 5: The algorithm used for backward traversal.

v, C_v, into a LUT that implements the functionality of the cone.

4. 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 (Equation 3) and to estimate the fanout of each node under mapping (Equation 4). Table 4 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 4-LUTs and the total area flow seen at the primary outputs (af) for the mapping solutions produced by each iteration. Data for two versions of IMap are reported: the first uses only depth bounds (DB) and the second uses both depth bounds and fanout estimation (DB + FE). Both versions of IMap produce identical mapping results in the first iteration. Depth bounds and fanout 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 depth bounds 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 which are estimates of actual mapping area. However, the first version does not use fanout 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 fanout estimates, makes steady progress beyond the second iteration guided by increasingly accurate area-flow values.

5. RESULTS

The MCNC circuits [16] were used to study the performance of IMap's depth-oriented and area-oriented mapping modes.

	DB		*DB + FE*	
Iteration	*LUTs*	*af*	*LUTs*	*af*
1	5107	8207.27	5107	8207.27
2	4800	6026.07	4650	4591.57
3	4779	5930.34	4588	4472.88
4	4776	5927.66	4569	4510.6
5	4776	5927.66	4552	4531.77
6	4776	5927.66	4547	4538.2
7	4776	5927.66	4545	4524.06
8	4776	5927.66	4545	4534.39

Table 4: Effect of iteration on two versions of IMap: one that uses depth bounds (*DB*) only and another that uses both depth bounds and fanout estimates (*DB + FE*).

Each circuit was first synthesized (SIS's [17] `script.rugged`) and decomposed into a network of two-input gates (SIS's `speedup` command) before being technology mapped into both 4-LUTs ($K = 4$) and 5-LUTs ($K = 5$). Although 4-LUTs have the greatest area efficiency [14] and are the most common type of LUT present in modern FPGAs, the 5-LUT mapping problem is important because some architectures allow two 4-LUTs to be combined into a single 5-LUT [15].

Table 5 compares the area of IMap's depth-oriented mapping solutions to those produced by two other mappers, CutMap [9] and FlowMap-r0 [2]. All three mappers produce depth-optimal mapping solutions and contain heuristics that reduce the area of the depth-optimal mapping solutions. While IMap uses the edge-delay model, CutMap and FlowMap-r0 use the unit-delay model. To produce comparable results, IMap assumes that every edge in the input graph is of unit delay. The RASP package [18], which contains CutMap and FlowMap-r0, contains two postprocessing operations called mppack [19] and flowpack [1]. These postprocessing operations can be used after technology mapping is completed and can help reduce area even further. The results presented include the application of the these operations. However, FlowMap-r0 is the only mapper that sees the largest area benefit from the application of these postprocessing operations; the operations help reduce FlowMap-r0's area by 5% when $K = 4$ and 7.6% when $K = 5$. The other two mappers see less than 2% area benefit when the postprocessing operations are applied. Although the table highlights results obtained for the 20 largest MCNC circuits, the totals at the bottom are for the entire set of MCNC circuits (202 circuits). When $K = 4$, CutMap and FlowMap-r0 produce solutions that are 13.0% and 8.8% larger than the solutions produced by IMap, respectively. When $K = 5$, the area advantage enjoyed by IMap over the other two mappers increases; CutMap's solutions are 15.3% larger while FlowMap-r0's solutions are 14.3% larger.

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 [2, 10].

When IMap is used in its area-oriented mapping mode, it

	$K=4$				$K=5$			
Circuit	*Depth*	*IMap*	*CutMaP*	*FlowMap*	*Depth*	*IMap*	*CutMaP*	*FlowMap*
C6288	26	895	544	549	23	647	655	713
alu4	8	1035	1207	1203	7	862	1027	1068
apex2	9	1201	1406	1441	8	1000	1212	1250
apex4	7	1086	1114	1130	7	877	1073	1085
bigkey	4	1594	1482	1370	4	916	915	1143
clma	16	4542	5250	5060	13	3728	4504	4688
des	7	1224	1359	1299	6	967	1030	1071
diffeq	13	842	1023	937	10	765	799	818
dsip	5	1150	1374	1598	4	1143	921	921
elliptic	17	2094	2734	2201	15	1875	2482	2028
ex1010	9	2524	2660	2676	9	1891	2264	2342
ex5p	8	918	1048	1025	6	820	875	892
frisc	22	2255	2919	2316	17	1935	2502	2028
i10	15	776	889	809	12	699	820	722
misex3	8	1086	1260	1259	7	907	1068	1096
pdc	11	1872	2144	2085	10	1452	1898	1864
s38417	11	3645	3786	3781	9	3026	3252	3135
s38584.1	11	3678	4423	3864	9	2761	3295	2983
seq	8	1089	1308	1291	6	969	1107	1161
spla	9	1333	1565	1566	8	1080	1262	1280
Total		58617	66252	63771		47542	54839	54332
Ratio		1.0	1.130	1.088		1.0	1.153	1.143

Table 5: Comparing IMap to CutMap and FlowMap-r0 when performing depth-oriented mapping with $K=4$ and $K=5$.

finds a mapping solution that minimizes area-flow. Table 6 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. 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. When $K=4$, the duplication free mapping solutions are 8.4% larger than the area-flow mapping solutions, and when $K=5$, the duplication free mapping solutions are 15.9% larger. Remarkably, in none of the 202 MCNC circuits was the area-flow mapping solution larger than the duplication free mapping solution.

6. SCALABILITY OF IMAP

The execution times for IMap remain reasonable when mapping with K set to 4 or 5. On a Pentium III 1 GHz computer, when $K=4$, the entire set of MCNC circuits was mapped in 100 seconds. When $K=5$, the mapping time for the MCNC circuits rose to 30 minutes. For larger values of K, the generation of all K-feasible cones takes far too long, and pruning techniques such as those described in [10] will be required to reduce the number of K-feasible cones generated for each node.

7. SUMMARY

This paper 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, called area-flow, is an estimation of actual mapping area and can be optimized using a dynamic pro-

	$K=4$		$K=5$	
Circuit	*AFlow*	*DFree*	*AFlow*	*DFree*
C6288	1114	1413	1016	1384
alu4	1017	1053	840	896
apex2	1158	1228	962	1061
apex4	1008	1092	845	991
bigkey	1039	1275	696	1047
clma	4293	4562	3451	3898
des	1161	1221	906	1062
diffeq	839	904	744	828
dsip	1153	1164	916	931
elliptic	2086	2115	1895	1999
ex1010	2093	2296	1790	2091
ex5p	892	994	754	903
frisc	2252	2359	1933	2115
i10	754	852	636	746
misex3	1077	1139	891	986
pdc	1753	1852	1432	1580
s38417	3683	4083	3109	3573
s38584.1	3736	4076	2767	3397
seq	1079	1156	886	1001
spla	1277	1361	1065	1191
Total	56370	61109	45960	53284
Ratio	1.0	1.084	1.0	1.159

Table 6: Comparing the optimal area flow mapping solution to the area-optimal duplication free mapping.

gramming 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 8.1% and 13.3% smaller than solutions produced by CutMap and FlowMap-r0. When mapping for area, IMap produced solutions that were between 7.8% and 13.7% smaller than the optimal duplication-free mapping solutions.

8. REFERENCES

[1] J. Cong and Y. Ding. An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. *IEEE Transactions on Computer-Aided Design*, Vol. 13, No. 1, January 1994, pp. 1–13.

[2] J. Cong and Y. Ding. On Area/Depth Tradeoff in LUT-Based FPGA Technology Mapping. *IEEE Transactions on VLSI Systems*, Vol. 2, No. 2, June 1994, pp. 137–148.

[3] Y. Kukimoto, R. K. Brayton and P. Sawkar. Delay-Optimal Technology Mapping by DAG Covering. In *Proceedings of the Design Automation Conference*, San Francisco, CA, June 1998, pp. 348–351.

[4] I. Levin and R. Y. Pinter. Realizing Expression Graphs using Table-Lookup FPGAs. In *Proceedings of the European Design Automation Conference*, Hamburg, Germany, September 1993, pp. 306–311.

[5] A. Farrahi and M. Sarrafzadeh. Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping. *IEEE Transactions on Computer-Aided Design*, Vol. 13, No. 11, pp. 1319–1332.

[6] K. Keutzer. DAGON: Technology Binding and Local Optimization by DAG Matching. In *Proceedings of the Design Automation Conference*, Miami Beach, FL, 1987, pp. 341–347.

[7] R. J. Francis, J. Rose and Z. G. Vranesic. Technology Mapping of Lookup Table-Based FPGAs for Performance. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, November 1991, pp. 568–571.

[8] R. J. Francis, J. Rose and Z. G. Vranesic. Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs. In *Proceedings of the ACM/IEEE Design Automation Conference*, San Francisco, CA, June 1991, pp. 227–233.

[9] J. Cong and Y. -Y. Hwang. Simultaneous Depth and Area Minimization in LUT-Based FGPA Mapping. In *Proceedings of the ACM International Symposium on FPGAs*, Monterey, CA, February 1995, pp. 68–74.

[10] J. Cong, C. Wu and Y. Ding. Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution. In *Proceedings of the ACM International Symposium on FPGAs*, Monterey, CA, February 1999, pp. 29–35.

[11] H. Yang and D. F. Wong. Edge-Map: Optimal Performance Driven Technology Mapping for Iterative LUT Based FPGA Designs. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, San Jose, CA, November 1994, pp. 150–155.

[12] J. Y. Lin, A. Jagannathan and J. Cong. Placement-Driven Technology Mapping for LUT-Based FPGAs. In *Proceedings of the ACM International Symposium on FPGAs*, Montery, CA, February 2003, pp. 121–126.

[13] M. Schlag, J. Kong and P. K. Chan. Routability-Driven Technology Mapping for Lookup Table-Based FPGAs. *IEEE Transactions on Computer-Aided Design*, Vol. 13, No. 1, pp. 13–26.

[14] 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 Journal of Solid-State Circuits*, Vol. 25, No. 5, October 1990, pp. 1217–1225.

[15] *Virtex II Platform FPGA Data Sheet*, Xilinx Inc., 2002. Available from `http://www.xilinx.com/apps/virtexapp.htm`.

[16] Collaborative Benchmarking Laboratory. *LGSynth93 Benchmark Suite*. Available from `http://www.cbl.ncsu.edu/www/`.

[17] 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*. Technical Report, University of California at Berkeley, 1992, Memorandum No. UCB/ERL M92/41.

[18] J. Cong, J. Peck and Y. Ding. RASP: A General Logic Synthesis System for SRAM-based FPGAs. In *Proceedings of the ACM International Symposium on FPGAs*, Montery, CA, February 1996, pp. 137–143.

[19] K. C. Chen, J. Cong, Y. Ding, A. Kahng and P. Trajmar. DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization. *IEEE Design and Test of Computers*, Vol. 9, No. 3, September 1992, pp. 7–20.

[20] A. Lu, G. Stenz and F. M. Johannes. Technology Mapping for Minimizing Gate and Routing Area. In *Proceedings of the Conference on Design, Automation and Test in Europe*, Le Palais des Congres de Paris, France, 1998, pp. 664–669.