Improvements to Technology Mapping for LUT-Based FPGAs

Alan Mishchenko, *Member, IEEE*, Satrajit Chatterjee, and Robert K. Brayton, *Fellow, IEEE*

***Abstract*—This paper presents several orthogonal improvements to the state-of-the-art lookup table (LUT)-based field-programmable gate array (FPGA) technology mapping. The improvements target the delay and area of technology mapping as well as the runtime and memory requirements. 1) Improved cut enumeration computes all K-feasible cuts, without pruning, for up to seven inputs for the largest Microelectronics Center of North Carolina benchmarks. A new technique for on-the-fly cut dropping reduces, by orders of magnitude, the memory needed to represent cuts for large designs. 2) The notion of cut factorization is introduced, in which one computes a subset of cuts for a node and generates other cuts from that subset as needed. Two cut factorization schemes are presented, and a new algorithm that uses cut factorization for delay-oriented mapping for FPGAs with large LUTs is proposed. 3) Improved area recovery leads to mappings with the area, on average, 6% smaller than the previous best work while preserving the delay optimality when starting from the same optimized netlists. 4) Lossless synthesis accumulates alternative circuit structures seen during logic optimization. Extending the mapper to use structural choices reduces the delay, on average, by 6% and the area by 12%, compared with the previous work, while increasing the runtime 1.6 times. Performing five iterations of mapping with choices reduces the delay by 10% and the area by 19% while increasing the runtime eight times. These improvements, on top of the state-of-the-art methods for LUT mapping, are available in the package ABC.**

***Index Terms*—Algorithms, area recovery, cut enumeration, field-programmable gate array (FPGA), lossless synthesis, technology mapping.**

I. Introduction

FIELD-PROGRAMMABLE gate arrays (FPGAs) are an attractive hardware design option, making technology mapping for FPGAs an important electronic design automation problem. For an excellent overview of the classical and recent work on FPGA technology mapping, focusing on area, delay, and power minimization, the reader is referred to [4].

Recent advanced algorithms for FPGA mapping, such as [4], [14], and [18], focus on area minimization under delay constraints. If delay constraints are not given, the optimum delay for the given logic structure is found first, and then the area is minimized without changing the delay.

In terms of the algorithms employed, mappers are divided into structural and functional. Structural mappers consider the circuit graph as given and find a covering of the graph with K-input subgraphs corresponding to lookup tables (LUTs). Functional approaches perform Boolean decomposition of the logic functions of the nodes into subfunctions of limited support size realizable by individual LUTs.

Since functional mappers explore a larger solution space, they tend to be time consuming, which limits their use to small designs. Thus, FPGA mapping for large designs is done using structural mappers, while functional mappers are used for resynthesis after technology mapping.

In this paper, we consider the recent work on DAOmap [4] as representative of the best structural technology mapping for LUT-based FPGAs. We refer to it as the previous work and discuss several ways of improving it. The improvements concern area and delay of the resulting LUT networks and the runtime and memory requirements of technology mapping. Specifically, our contributions fall into three categories.

A. *Improved Cut Computation*

Computation of all K-feasible cuts is typically a runtime and memory bottleneck of a structural mapper. We propose several enhancements to the standard cut enumeration procedure [23], [9]. Specifically, we introduce cut filtering with signatures and show that it leads to a speed up. This makes exhaustive cut enumeration for six and seven inputs practical for many test cases.

Since the number of K-feasible cuts, for large K, can exceed 100 per node, storing all the computed cuts in memory is problematic for large benchmarks. We address this difficulty by allowing cut enumeration to drop those cuts at the nodes whose fanouts have already been processed. This allows the mapper to store only a small fraction of all K-feasible cuts at any time, thereby reducing memory usage for large benchmarks by an order of magnitude or more.

B. *Using Factor Cuts*

Enumerating K-feasible cuts for a large K becomes important when FPGA mapping targets macrocells. Such cells are typically composed of LUTs, multiplexers, and elementary gates and can implement a subset of functions of K inputs. Although our improved cut enumeration efficiently computes all cuts up to seven inputs, it is not practical for 8–12 inputs,

Manuscript received March 16, 2006; revised July 20, 2006. This work was supported in part by the NSF under Contract CCR-0312676, in part by the MARCO Focus Center for Circuit System Solution under Contract 2003-CT-888, and in part by the California Micro program with industrial sponsors, Altera, Intel, Magma, and Synplicity. This paper was recommended by Associate Editor K. Bazargan.

The authors are with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720 USA (e-mail: alanmi@eecs.berkeley.edu; satrajit@eecs.berkeley.edu; brayton@eecs.berkeley.edu).

Digital Object Identifier 10.1109/TCAD.2006.887925

0278-0070/$25.00 © 2007 IEEE

which is the size typical for most macrocells, simply because there are too many cuts.

Since only a very small fraction of all cuts of large size is used in FPGA mapping, different heuristics have been proposed to prune the cuts, for example, [9]. The problem is that the delay optimality for large cut sizes is not guaranteed; in practice, their deviation from the optimum delay for the given logic structure may be substantial. This is because heuristics often prune cuts that are not optimal for a node but may lead to optimal cuts of the fanouts.

To address the enumeration problem, we introduce the notion of cut factorization, which is conceptually related to the notion of disjoint-support decomposition of logic functions [2]. Just as the algebraic expression $(ab + ac)$ can be factored as $a(b + c)$, the set of all cuts of a node can be factored using two sets of cuts called *global* and *local.* Collectively, they are called *factor cuts.* By expanding factor cuts w.r.t. local cuts, a larger set of cuts can be obtained. During the cut computation, only factor cuts are enumerated. Later on, during mapping, other cuts are generated from factor cuts as necessary.

Depending on how global and local cuts are defined, there can be different factorization schemes. In this paper, we present two schemes, namely: 1) complete and 2) partial. In complete factorization (CF), every cut can be obtained by expanding a factor cut w.r.t. a local cut. However, CF is expensive since there may be a large number of global cuts.

Partial factorization (PF) is an alternative approach where there are much fewer global cuts, but there is no guarantee that all cuts can be generated by expanding factor cuts. However, in practice, good cuts are obtained with PF in a fraction of the runtime required for complete enumeration.

C. Better, Simpler, and Faster Area Recovery

Area optimization after delay-optimum structural mapping proceeds in several passes over the network. Each pass assigns cuts with a better area among the ones that do not violate the required time. Previous work [4] relied on several sophisticated heuristics for ranking the cuts, trying to estimate their potential to save area. They concluded that although the heuristics are not equally useful, to get good area, a number of them need to be applied.

In this paper, we show that the combination of two simple techniques is enough to improve the area results of the previous work by 6% on average while achieving the optimum delay. The proposed combination is synergistic since the first one attempts heuristically to find a global optimum, whereas the second ensures that at least a local optimum is reached.

It should be noted that the first heuristic (known as *effective area* [9] or *area flow* [18]) was used in the previous work but applied in a reverse topological order while we argue that a forward topological order works better.

D. Lossless Synthesis

The main drawback of the structural approaches to technology mapping is their dependence on the initial circuit structure (called structural bias). If the structure is bad, neither heuristics nor iterative area recovery will improve the results of mapping.

To obtain a good structure for the network, usually several technologically independent synthesis steps are performed. An example is *script.rugged* in SIS followed by a two-input gate decomposition. Each synthesis step in the script is heuristic, and the subject graph produced at the end is not necessarily optimum. Indeed, it is possible that the initial or an intermediate network is better in some respects than the final network.

In this paper, we explore the idea of combining these intermediate networks into a single subject graph with choices, which is then used to derive the mapped netlist. Thus, the mapper is not constrained to use any one network but can pick the best parts of each. We call this approach *lossless synthesis* since no network seen during the synthesis process is ever lost. By including the initial network in the choice network, the heuristic logic synthesis operations can never make things worse. Also, multiple scripts can be used to accumulate more choices. We defer discussion of related work to Section VI-C.

In summary, we note that the contributions are largely orthogonal in nature and tend to reinforce each other. For example, improved cut enumeration gives extra speed to the computation of factor cuts and cuts for the networks with choices. Similarly, the proposed area recovery heuristics will lead to an even smaller area when factor cuts are used. However, the interaction of factor cuts and lossless synthesis is less obviously beneficial. Investigation of this issue is deferred to future work.

The rest of this paper is organized as follows: Section II describes the background. Sections III–VI give details on the four contributions of this paper listed. Section VII shows the experimental results. Section VIII concludes this paper and outlines future work.

II. Background

A *Boolean network* is a directed acyclic graph (DAG) with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates. The terms network, Boolean network, and circuit are used interchangeably in this paper.

A node has zero or more *fanins,* i.e., nodes that are driving this node, and zero or more *fanouts,* i.e., nodes driven by this node. The *primary inputs* (PIs) of the network are nodes without fanins in the current network. The *primary outputs* (POs) are a subset of nodes of the network. If the network is sequential, the flip-flop outputs/inputs are treated as additional PIs/POs. In the following, it is assumed that each node has a unique integer number called the *node ID.*

A network is K *bounded* if the number of fanins of each node does not exceed K. A *subject graph* is a K-bounded network used for technology mapping. Any combinational network can be represented as an AND–INV graph (AIG) composed of two-input AND and inverters. Without limiting the generality, in this paper, we assume subject graphs to be AIGs.

A *cut* C of node n is a set of nodes of the network, called *leaves,* such that each path from a PI to n passes through at least one leaf. A *trivial cut* of a node is the cut composed of the node itself. A cut is K *feasible* if the number of nodes in it does not exceed K. A cut is said to be *dominated* if it contains another cut of the same node.

A *fanin* (*fanout*) *cone* of node n is a subset of all nodes of the network reachable through the fanin (fanout) edges from the given node. A *maximum fanout free cone* (MFFC) of node n is a subset of the fanin cone such that every path from a node in the subset to the POs passes through n. Informally, the MFFC of a node contains all the logic used only by the node. Thus, when a node is removed or substituted, the logic in its MFFC can also be removed.

The *level* of a node is the length of the longest path from any PI to the node. The node itself is counted in the path lengths but the PIs are not counted. The network *depth* is the largest level of an internal node in the network. The delay and area of an FPGA mapping are measured by the depth and number of LUTs in the resulting LUT network.

A typical procedure for structural technology mapping consists of the following steps:

1) cut computation;
2) delay-optimum mapping;
3) area recovery using heuristics;
4) recording the resulting LUT network.

For a detailed description on these steps, we refer the reader to [4] and [18].

III. Improved Cut Computation

Structural technology mapping into K-input LUTs starts by computing K-feasible cuts for each internal two-input node of the subject graph. The number of K-feasible cuts of a network with n nodes is $O(n^K)$ [8].

In this section, we focus on improving the implementation of the cut computation. The asymptotic complexity of the cut computation procedures is still quadratic in the number of cuts stored at a node, but the improvements make the algorithm faster in practice, applicable to larger circuits, and scalable to larger values of K.

A. Cut Enumeration

We begin with a review of the standard procedure for enumerating, for each node of an AIG, the set of all of its K-feasible cuts [9], [23]. Let A and B be two sets of cuts. For convenience, we define the operation $A\Diamond B$ as

$$A\Diamond B = \{u \cup v | u \in A, v \in B, |u \cup v| \leq k\}.$$

Let $\Phi(n)$ denote the set of K-feasible cuts of node n. If n is an AND node, let n_1 and n_2 denote its fanins. We have

$$\Phi(n) = \left\{ \begin{array}{ll} \{\{n\}\} & : n \in \text{PI} \\ \{\{n\}\} \cup [\Phi(n_1) \Diamond \Phi(n_2)] & : \text{otherwise} \end{array} \right\}.$$

This formula translates into a simple procedure that computes all K-feasible cuts in a single pass from the PIs to the POs in a topological order. Informally, the cut set of an AND node is computed by merging the two cut sets of its children and adding the trivial cut (the node itself). This is done by taking the pairwise unions of cuts belonging to the fanins while keeping only K-feasible cuts.

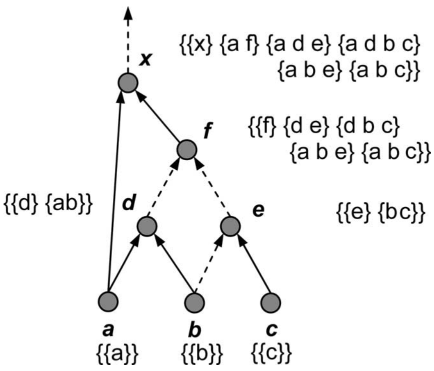

Fig. 1. Illustration of cut computation.

In this process of merging the cut sets to form the resulting cut set, it is necessary to detect duplicate cuts and remove dominated cuts. Removing them before computing cuts for the next node reduces the number of cut pairs considered without impacting the quality of mapping. In practice, the total number of cut pairs tried during the merging greatly exceeds the number of K-feasible cuts found. This makes checking the K feasibility of the unions of cut pairs and testing duplication and dominance of individual cuts the performance bottleneck of the cut computation.

Example: Fig. 1 illustrates the bottom-up cut enumeration procedure for a small circuit. Observe that due to reconvergence, the cut set of node x contains a dominated cut $\{a\, d\, b\, c\}$ (dominated by $\{a\, b\, c\}$) that may be removed without affecting the quality of mapping.

B. Using Signatures

In this paper, we propose to use signatures for testing cut properties, such as duplication, dominance, and K feasibility. Conceptually, it is similar to the use of Bloom filters for encoding sets [3] and to the use of signatures for comparing clauses in [11]. The use of signatures only speeds up the computation; no additional pruning is done.

A signature sign(C) of cut C is an M-bit integer whose bitwise representation contains 1s in the positions corresponding to the node IDs. The signature is computed by the bit-wise OR of integers as

$$\text{sign}(C) = \sum_{n \in C} 2^{\text{ID}(n) \bmod M}.$$

Testing cut properties with signatures is much faster than testing them by directly comparing leaves. The following propositions state necessary conditions for duplication, dominance, and K feasibility of cuts. If these conditions are violated, then there is no need to do a detailed check by comparing leaves. If the conditions hold, then a detailed check is done to establish the property. (The detailed test cannot be avoided due to aliasing: two different cuts may have the same signature).

Proposition 1: If cuts C_1 and C_2 are equal, so are their signatures.

Proposition 2: If cut C_1 dominates cut C_2, the 1s of $\mathrm{sign}(C_1)$ are contained in the 1s of $\mathrm{sign}(C_2)$.

Proposition 3: If $C_1 \cup C_2$ is a K-feasible cut, $|\mathrm{sign}(C_1) + \mathrm{sign}(C_2)| \leq K$. Here, $|n|$ denotes the number of 1s in the binary representation of n, and addition is the bitwise OR.

Our current implementation uses one machine word (composed of 32 bits on a 32-bit machine) to represent the signature of a cut, i.e., $M = 32$. As a result, most of the checks are performed using several bit-wise machine operations, and only if the signatures fail to disprove a property is the actual comparison of leaves performed.

Example: Let $M = 8$ (for ease of exposition). The cut C_1 with nodes having IDs 32, 68, and 69 would have $\mathrm{sign}(C_1) = 00110001$. A second cut C_2 with nodes having IDs 32, 68, and 70 would have $\mathrm{sign}(C_2) = 01010001$. From comparing the two signatures, it is clear that neither C_1 dominates C_2 or vice-versa. Thus, there is no need to examine the leaves of C_1 and C_2 to establish dominance. Let C_3 be a third cut with node IDs 36, 64, and 69. Now $\mathrm{sign}(C_3) = 00110001 = \mathrm{sign}(C_1)$. However, C_3 is *not* equal to C_1. (Thus, to establish properties, a comparison of the cut leaves is necessary.)

C. Practical Observations

In the literature on technology mapping, all four-input and five-input cuts are typically computed exhaustively, whereas computation of cuts with more inputs is considered time consuming because of the large number of these cuts. Different heuristics have been proposed [9] to rank and prune cuts to reduce the runtime. We experimented with these heuristics and found that they are effective for area but lead to suboptimal delay.

In order to preserve delay optimality, we focus on perfecting the cut computation and computing all cuts whenever possible. Pruning is done only if the number of cuts at a node exceeds a predefined limit set to 1000 in our experiments. When computing K-feasible cuts with $4 \leq K \leq 7$ for the largest Microelectronics Center of North Carolina (MCNC) benchmarks, this limit was never reached, and hence no pruning was performed, meaning that the cuts were computed exhaustively. Due to the use of signatures, the runtime for $4 \leq K \leq 7$ was also quite affordable, as evidenced by the experiments. However, for eight-input cuts, pruning was required for some benchmarks.

D. Reducing Memory for Cut Representation

The number of K-feasible cuts for $K > 5$ can be large. The average number of exhaustively computed seven-input cuts in the largest MCNC benchmarks is around 95 cuts per node. In large industrial designs, the total number of cuts could be of the order of tens of millions. Therefore, once the speed of cut enumeration is improved, memory usage for cut representation becomes the next pressing issue.

To address this issue, we modified the cut enumeration algorithm to free the cuts as soon as they are not needed for the subsequent enumeration steps. This idea is based on the observation that the cuts of the nodes, whose fanouts have already been processed, can be deallocated without impacting cut enumeration. It should be noted that if technology mapping is performed in several topological passes over the subject graph, the cuts are recomputed in each pass. However, given the speed of the improved cut computation, this does not seem to be a problem.

Experimental results (presented in Table II) show that by enabling cut dropping, as explained, the memory usage for the cut representation is reduced by an order of magnitude for MCNC benchmarks. We see that for larger benchmarks, the reduction in memory is even more substantial.

It is possible to reduce the runtime of the repeated cut computation by recording the *cut enumeration trace,* which is saved during the first pass of cut enumeration and used in subsequent passes. The idea is based on the observation that, even when signatures are used, the most time-consuming part of cut enumeration is determining what cut pairs lead to nonduplicated nondominated K-feasible cuts at each node. The number of such cut pairs is very small compared with the total number of cut pairs at each node. The cut enumeration trace recorded in the first pass compactly stores information about all such pairs and the order of merging them to produce all the K-feasible cuts at each node. The trace serves as an oracle for the subsequent cut enumeration passes, which can now skip checking all cut pairs and immediately derive useful cuts.

This option was implemented and tested in our cut enumeration package but was not used in the experimental results because the benchmarks allowed for storing all the cuts in memory at the same time. We mention this option here because we expect it to be useful for industrial mappers working on very large designs.

IV. Factor Cuts

This section introduces the notion of *cut factorization* to address the problem of cut enumeration. In cut factorization, we identify certain subsets of the set of cuts of a node—the *local* cuts and the *global* cuts, collectively called *factor* cuts—and use these to generate the other cuts when needed. Depending on how local and global cuts are defined, we get different schemes for factorization. In this work, we consider two schemes, namely: 1) *complete* and 2) *partial.* In CF, all cuts can be derived from factor cuts, but it is expensive (although less so than complete enumeration). In PF, not all cuts can be generated from factor cuts, but it is very fast in practice and produces good results.

In this section, we present the theory of cut factorization and consider a sample application to compute delay optimal FPGA mapping for large LUTs. For most nodes in a network, examining only factor cuts is enough to achieve the optimum delay. For the remaining few nodes, a small number of nonfactor cuts have to be considered.

A. Preliminaries

1) Dag and Tree Nodes: Consider an AIG G. A *dag node* is a node of G that has two or more outgoing edges. A node of G

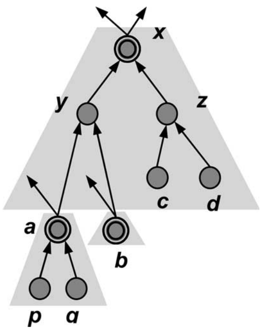

Fig. 2. AIG fragment to illustrate cut factorization.

that is not a dag node is called a *tree node*. The set of dag nodes is denoted by D and the tree nodes by T.

The subgraph G_T of G induced by the tree nodes is a forest of trees. Each tree T in G_T has an outgoing edge to exactly one DAG node n_d in G.

Consider the subgraph T_{nd} of G induced by a DAG node n_d in G and the nodes belonging to the trees in G_T that feed into it. T_{nd} is a (possibly trivial) tree. T_{nd} is called the *factor tree* of a node n in T_{nd}. Clearly, every node in G has a factor tree. The DAG node n_d is called the root of T_{nd}.

The leaves n_i of a factor tree are dag nodes. The factor tree along with the inputs n_i is a leaf-DAG and is called the *factor leaf-dag*. Every node n in G has a unique factor leaf-dag (via its unique factor tree). The root of a factor leaf-dag is the root of the corresponding tree.

Example: Consider the AIG shown in Fig. 2. Nodes p, q, b, c, and d are PIs. Nodes, such as x and a, that have double circles are dag nodes. The rest are tree nodes. The set of nodes in each shaded region forms a factor tree. The factor tree for node b is trivial. The factor tree of node x consists of x, y, z, c, and d. The factor leaf-dag of x contains the nodes in factor tree of x along with nodes a and b.

Local Cuts, Global Cuts, and Expansion: In the following sections, we will identify some K-feasible cuts in the network as local cuts and some others as global cuts. We refer to them collectively as factor cuts. The precise definitions of local and global will depend on the factorization scheme (complete or partial), but the general idea is to *expand* factor cuts by local cuts to obtain other K-feasible cuts. In the case of CF, this expansion will produce all K-feasible cuts.

Let c be a factor cut of node $n \in G$. Let c_i be a local cut of a node $i \in c$. Consider $l = (\bigcup_i c_i)$. l is a cut of n although not necessarily K feasible. If l is K feasible, then l is called a *one-step expansion* of c. Define 1-step(c) as the set of cuts obtained from c by one-step expansion, i.e.,

$$\text{1-step}(c) = \{l \mid l \text{ is a one-step expansion of } c\}.$$

We ensure that $c \in \text{1-step}(c)$ by requiring that every node have the trivial cut as a local cut.

Example: In Fig. 2, consider the cut $\{a, b, z\}$ of x. By expanding node a with its local cut $\{p, q\}$, we obtain the cut $\{p, q, b, z\}$ of x. Thus, $\{p, q, b, z\} \in \text{1-step}(\{a, b, z\})$.

B. Complete Factorization (CF)

In CF, we enumerate *tree* cuts and *reduced* cuts (defined in the following), which are subsets of the set of all K-feasible cuts. Tree cuts are the local cuts, and reduced cuts are the global cuts. We use the term *complete factor cuts* to refer to tree cuts and reduced cuts collectively. CF has the property that any K-feasible cut can be obtained by one-step expansion.

Tree Cuts (Local Cuts): Let $\Phi_T(n)$ denote the set of all tree cuts of node n. First, define the auxiliary function $\Phi_T^{\dagger}(n)$ as

$$\Phi_T^{\dagger}(n) = \left\{ \begin{array}{c} \varnothing : n \in D \\ \Phi_T(n) : \text{otherwise} \end{array} \right\}.$$

Now, $\Phi_T(n)$ is defined recursively as

$$\Phi_T(n) = \left\{ \begin{array}{ll} \{\{n\}\} & : n \in \text{PI} \\ \{\{n\}\} \cup \Phi_T^{\dagger}(n_1) \Diamond \Phi_T^{\dagger}(n_2) & : \text{otherwise} \end{array} \right\}.$$

$\Phi_T(n)$ represents the subset of K-feasible cuts of n that only involve nodes from the factor tree of n.

Example: In Fig. 2, $\Phi_T(x) = \{\{x\}, \{y, z\}, \{y, c, d\}\}$.

Reduced Cuts (Global Cuts): We define $\Phi_R(n)$, the set of reduced cuts of a node n, as

$$\Phi_R(n) = \left\{ \begin{array}{ll} \{\{n\}\} & : n \in \text{PI} \\ \{\{n\}\} \cup ((\Phi_R(n_1) \Diamond \Phi_R(n_2)) \backslash \Phi_T(n)) & : \text{otherwise} \end{array} \right\}.$$

The formula for $\Phi_R(n)$ is very similar to that of $\Phi(n)$ except that nontrivial tree cuts are removed. Since this removal is done recursively, $\Phi_R(n)$ is significantly smaller than $\Phi(n)$.

Example: In Fig. 2, $\Phi_R(x) = \{\{x\}, \{a, b, z\}\}$. Note that $\{a, b, c, d\}$ is not a reduced cut of x since $\{c, d\}$ is removed when computing $\Phi_R(z)$.

Cut Decomposition Theorem: With local and global cuts being tree and reduced cuts, respectively, a cut decomposition theorem holds.

Theorem 1: Every K-feasible cut of node n in G is a one-step expansion of a K-feasible complete factor cut of n, i.e., if $c \in \Phi(n)$, then $\exists c' \in \Phi_R(n)$ s.t. $c \in \text{1-step}(c')$.

Proof Sketch: Let c be a K-feasible cut of n. If c consists of nodes only from the factor tree T_n of n, then c is a local cut of n and $c \in \text{1-step}(\{n\})$, and the theorem is proved.

Suppose c has some nodes $\{n_i\} \subset c$ belonging to a different factor tree T whose root is x. Consider the set $c' = c \setminus \{n_i\} \cup \{x\}$. Node c' is also a cut of x since every path through $\{n_i\}$ passes through x. Furthermore, c' is K feasible since $|c'| \leq |c|$ by construction. Now, $c' \in \Phi_R(n)$ since n is a DAG node and $c \in \text{1-step}(c')$.

If c has nodes from multiple factor trees, a similar argument holds. ■

C. Partial Factorization (PF)

Although CF causes a reduction in the number of cuts that need to be enumerated, further reduction is possible by sacrificing the ability to generate all K-feasible cuts by one-step expansion. This leads to the notion of PF. PF is much faster than CF and produces a *good* set of cuts in practice, especially for large K (say $K = 9$).

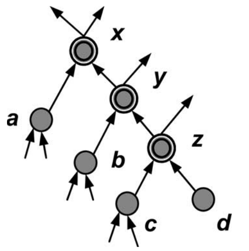

Fig. 3. Example of a limitation of PF.

In PF, *leaf-dag* cuts play the role of local cuts, and *dag* cuts play the role of global cuts. We use the term *partial factor cuts* to refer to leaf-dag cuts and dag cuts collectively.

1) Leaf-Dag Cuts (Local Cuts): Let $\Phi_L(n)$ denote the set of K-feasible leaf-dag cuts of node n. Define the auxiliary function $\Phi^{\dagger}_L(n)$ as

$$\Phi^{\dagger}_L(n) = \left\{ \begin{array}{c} \{\{n\}\} : n \in D \\ \Phi_L(n) : \text{otherwise} \end{array} \right\}.$$

Now, $\Phi_L(n)$ is defined recursively as

$$\Phi_L(n) = \left\{ \begin{array}{ll} \{\{n\}\} & : n \in \text{PI} \\ \{\{n\}\} \cup \left(\Phi^{\dagger}_L(n_1) \diamondsuit \Phi^{\dagger}_L(n_2)\right) & : \text{otherwise} \end{array} \right\}.$$

$\Phi_T(n)$ represents the subset of K-feasible cuts of n that only involve nodes from the factor tree of n.

Conceptually, leaf-dag cuts are similar to tree cuts. Unlike tree cuts, leaf-dag cuts also include the dag nodes that feed into the factor tree of a node. This allows more cuts to be generated by one-step expansion at the cost of a slight increase in runtime for local cut enumeration.

Example: In Fig. 2, the cuts $\{a, b, z\}$ and $\{a, b, c, d\}$ are examples of leaf-dag cuts of node x. (They are not tree cuts of x.)

Dag Cuts (Global Cuts): Let $\Phi_D(n)$ denote the set of K-feasible dag cuts of n. We define

$$\Phi_D(n) = \left\{ \begin{array}{ll} \{\{n\}\} & : n \in \text{PI} \\ \Phi_D(n_1) \diamondsuit \Phi_D(n_2) & : n \in T \\ \{\{n\}\} \cup (\Phi_D(n_1) \diamondsuit \Phi_D(n_2)) & : \text{otherwise} \end{array} \right\}.$$

Example: In Fig. 2, for $K = 4$, $\{x\}$ and $\{a, b, c, d\}$ are the only dag cuts of x.

This definition of dag cuts is motivated by a need to reduce the number of global cuts seen in CF. Defining dag cuts in this manner allows us to capture much of the reconvergence in the network without having to enumerate the large number of reduced cuts (as in CF).

However, by computing global cuts this way, some cuts cannot be generated by one-step expansion, as shown in Fig. 3. The four-feasible cut $\{a, b, c, d\}$ of x cannot be generated using one-step expansion of a partial factor cut of x.

D. Delay-Optimum K-LUT Mapping

In this section, we apply factor cuts to technology mapping for FPGAs with large LUTs. Most of the present-day FPGA architectures do not provide LUTs of size more than 6. Instead, they contain macrocells, which can implement a subset of functions with 8–12 inputs. The algorithm presented in this section is only used to illustrate the use of factor cuts. The extension of the proposed algorithm to macrocells is left for future research.

The conventional algorithm for delay optimal K-LUT mapping enumerates all K-feasible cuts and chooses the best set of cuts using dynamic programming on the AIG. The algorithm proceeds in two passes over the nodes.

The first pass, called the forward pass, is in topological order from PIs to POs. For each node, all K-feasible cuts are enumerated (using the K-feasible cuts of its children), and the cut with the earliest arrival time is selected.

The arrival time of a cut c, denoted by arrival(c), is defined as

$$\text{arrival}(c) = 1 + \max_{n \in c} \text{arrival}(n)$$

where arrival(n) is the best arrival time for node n (from among all its K-feasible cuts). This recursion is well defined, since when the cuts for a node n are being processed, the nodes in the transitive fanin of n have already been processed. Thus, the best arrival time of any node in a K-feasible cut of n has already been computed.

The second pass of the algorithm is done in reverse topological order. For each PO, the best K-feasible cut is chosen, and an LUT is constructed in the mapped netlist to implement it. Then, recursively for each node in this best cut, this procedure is repeated.

The main limitation of the conventional algorithm is that it explicitly enumerates a large number of all K-feasible cuts during the forward pass. The idea behind using factor cuts is to avoid this enumeration. Ideally, one would like to enumerate only factor cuts, which are far fewer than K-feasible cuts. However, there is no guarantee that the best K-feasible cut is a factor cut. To avoid all possible one-step expansions, which could be as bad as enumerating all K-feasible cuts, we use Lemma 2 from [7].

Theorem 2 [7]: In Algorithm 1, if n is an AND node with inputs n_1 and n_2, then arrival$(n) = p$ or arrival$(n) = p + 1$, where $p = \max(\text{arrival}(n_1), \text{arrival}(n_2))$.

Theorem 2 provides a lower bound on the best arrival time. If a factor cut attains the lower bound, then no one-step expansions are necessary. If no factor cut attains the lower bound, then they are one-step expanded one by one. During this process, if the lower bound is attained, further expansion is not needed.

Optimality: If CF is used, then this algorithm produces the optimal delay since one-step expansion will produce all K-feasible cuts (by the cut decomposition theorem). In the case of PF, there is no guarantee of optimality. However, experiments show that for large K, there is no loss of optimality for the set of benchmarks considered (see Section VII-D).

Expansion: In CF, one-step expansion need not be exhaustive. It suffices to expand the late arriving inputs of the cut, one node at a time. This is because the expansions are independent—two nodes in the cut do not have to be expanded simultaneously with their tree cuts since the tree cuts of two nodes never overlap.

In PF, the leaf-dag cuts of two nodes may overlap, and so the expansions are not independent. However, in our experiments, the late-arriving nodes were expanded one node at a time since that did not degrade the quality significantly.

It is instructive to see why the conventional algorithm cannot be easily modified to exploit the lower bound. Although one need not scan all of $\Phi(n)$ to find the best cut (one can stop as soon as the lower bound is attained), one still needs to construct $\Phi(n)$ completely. This is because a cut $c \in \Phi(n)$ that does not lead to the best arrival time for n may lead to the best cut for some node n' in the transitive fanout of n.

V. Improved Area Recovery

Exact area minimization during technology mapping for DAGs is NP-hard [12] and hence not tractable for large circuits. Various heuristics for approximate area minimization during mapping have shown good results [4], [14], [18].

In this paper, we use a combination of only two heuristics, which work well in practice. The order of applying the heuristics is important since they are complementary. The first heuristic has a global view and selects logic cones with more shared logic. The second heuristic provides a missing local view by minimizing the area exactly at each node.

A. Global View Heuristic

Area flow [18] (*effective area* [9]) is a useful extension of the notion of area. It can be computed in one pass over the network from the PIs to the POs. Area flow for the PIs is set to 0. The area flow at node n is

$$\mathrm{AF}(n) = \left[\mathrm{Area}(n) + \Sigma_{\mathrm{i}}\mathrm{AF}\left(\mathrm{Leaf}_{\mathrm{i}}(n)\right)\right] / \mathrm{NumFanouts}(n)$$

where Area(n) is the area of the LUT used to map the current best cut of node n, Leaf$_i(n)$ is the ith leaf of the best cut at n, and NumFanouts(n) is the number of fanouts of node n in the currently selected mapping. If a node is not used in the current mapping, for the purposes of area flow computation, its fanout count is assumed to be 1.

If nodes are processed from the PIs to the POs, the computing area flow is fast. Area flow gives a global view of how useful the logic is in the cone for the current mapping. Area flow estimates sharing between cones without the need to retraverse them.

In our mapper, as in the previous work [4], [18], area flow is used as a tiebreaker in the first pass when a delay-optimum mapping is computed. In the first stage of area recovery, area flow becomes the primary cost function used to choose among the cuts, whose arrival times do not exceed the required times.

B. Local View Heuristic

The second heuristic providing a local view for area recovery in our mapper is not used in the previous work. This heuristic proceeds in topological order and looks at the exact local area to be gained by updating the best cut at each node. The *exact area of a cut* is defined as the sum of areas of the LUTs in the MFFC of the cut, i.e., the LUTs to be added to the mapping if the cut is selected as the best one.

The exact area of a cut is computed using a fast local Depth First Search (DFS) traversal of the subject graph starting from the root node of the cut. The *reference counter of a node* in the subject graph is equal to the number of times it is used in the current mapping, i.e., the number of times it appears as a leaf of the best cut at some other node or as a PO. The exact area computation procedure is called for a cut. It adds the cut area to the local area being computed, dereferences the cut leaves, and recursively calls itself for the best cuts of the leaves whose reference counters are zero. This procedure recurs as many times as there are LUTs in the MFFC of the cut, for which it is called. This number is typically small, which explains why computing the exact area is reasonably quick. Once the exact area is computed, a similar recursive referencing is performed to reset the reference counters to their initial values before computing the exact area for other cuts.

MFFCs were used in [8] for duplication-free mapping, which was alternated with depth relaxation for area minimization. This paper differs from [8] in that it is not restricted to duplication-free mapping but employs the concept of MFFC along with reference counting of nodes in the AIG for accurate estimation of the impact of cut selection on area during mapping.

Experimentally, we found that, after computing a delay-optimum mapping, two passes of area recovery are enough to produce good-quality mapping. The first pass uses area flow, and the second one uses the exact local area. Iterating area recovery using both of the heuristics can additionally save up to 2% of the total area of mapping, which may or may not justify the extra runtime.

It is interesting to observe that the previous work recovers area at each node in *reverse* topological order. We argue that the opposite works better for incremental area recovery since it allows most of the slack to be used on noncritical paths closer to the PIs, where the logic is typically wider and hence offers more opportunity for area savings.

VI. Lossless Synthesis

The idea behind lossless logic synthesis is to *remember* some or all networks seen during a logic synthesis flow (or a set of flows) and to select the best parts of each network during technology mapping. This is useful for two reasons.

First, technology-independent synthesis algorithms are heuristic, and so there is no guarantee that the final network is optimum. When only this final network is used, the mapper may miss a better result that could be obtained from part of an intermediate network in the flow.

Second, synthesis operations usually apply a cost function (e.g., delay) to the network as a whole. Thus, a flow to optimize delay may significantly increase the area. However, by combining a delay-optimized network with one optimized for area, it is possible to get the best of both; on the critical path, the mapper can choose from the delay-optimized network, off critical from the area-optimized network, and near critical from both.

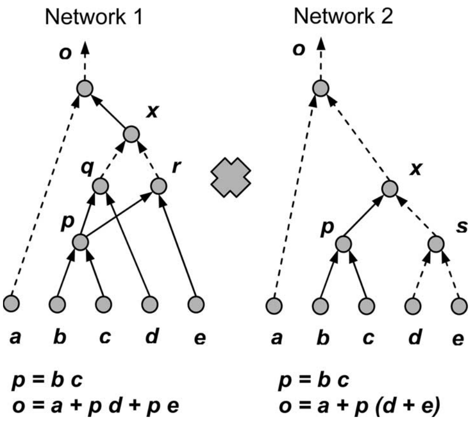

Fig. 4. Equivalent networks before choicing.

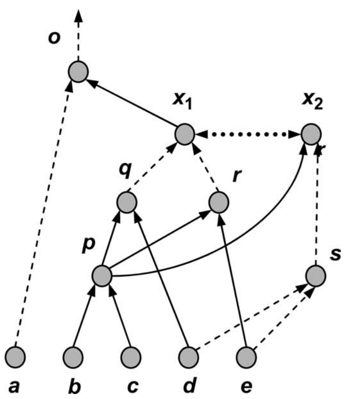

Fig. 5. Choice network.

Section VI-A gives an overview of constructing the choice network efficiently. Section VI-B extends the cut computation to handle choices.

A. Constructing the Choice Network

The choice network is constructed from a collection of networks that are functionally equivalent. The identification of functionally equivalent nodes has been a key component in recent advances in equivalence checking [15], [17].

Conceptually, the procedure is as follows: Each network is decomposed into an AIG. All the nodes with the same global function in terms of the PIs are collected in equivalence classes. The result is a *choice AIG* that has multiple functionally equivalent points grouped together.

The identification of functionally equivalent points could be done by computing global Binary Decision Diagrams (BDDs), but this is not feasible for large circuits. One can use random simulation to identify potentially equivalent nodes and then use a Satisfiability (SAT) engine to verify equivalence and construct the equivalence classes. To this end, we implemented a package called functionally reduced AND inverter graphs. This package exposes APIs comparable to a BDD package but internally uses simulation and SAT. More details may be found in [19].

Example: Figs. 4 and 5 illustrate the construction of a network with choices. Networks 1 and 2 in Fig. 4 show the subject graphs obtained from two networks that are functionally equivalent but structurally different. The nodes x_1 and x_2 in the two subject graphs are functionally equivalent (up to complementation). They are combined in an equivalence class in the choice network, and an arbitrary member (x_1 in this case) is set as the class representative. Node p does not lead to a choice because p is structurally the same in both networks. Note also that there is no choice corresponding to the output node o since the procedure detects the maximal commonality of the two networks.

A different way of generating choices is by Λ and Δ transformations [16]. Given an AIG, the associativity of the AND operation is used to locally rewrite the graph (the Λ transformation), i.e., whenever the structure AND(AND(x_1, x_2), x_3) is seen in the AIG, it is replaced by the equivalent structures AND(AND(x_1, x_3), x_2) and AND(x_1, AND(x_2, x_3)). If this process is done until no new AND nodes are created, it is equivalent to identifying the maximal multi-input AND gates in the AIG and adding all possible tree decompositions of these gates. Similarly, the distributivity of AND over OR (the Δ transformation) provides another source of choices.

Using structural choices leads to a new way of thinking about logic synthesis: rather than trying to come up with a good final netlist used as an input to mapping, one can postpone decisions and simply accumulate choices by applying arbitrary transformations, which lead to improvement in some sense. The best combination of these choices is selected finally during mapping.

B. Cut Enumeration With Choices

The cut-based structural FPGA mapping procedure can be extended naturally to handle equivalence classes of nodes. It is remarkable that only the cut enumeration step needs modification.

Given a node n, let N denote its equivalence class. Let $\Phi(N)$ denote the set of cuts of the equivalence class N. Then, it is obvious that $\Phi(N) = \bigcup_{n \in N} \Phi(n)$. In addition, if a and b are the two inputs of n belonging to equivalence classes A and B, respectively, then

$$\Phi(n) = \{\{n\}\} \cup \{u \cup v | u \in \Phi(A), v \in \Phi(B), |u \cup v| \leq k\}.$$

This expression for $\Phi(n)$ is a slight modification of the one used in Section III to compute the cuts without choices. The cuts of n are obtained from the cuts of the *equivalence classes* of its fanins (instead of the cuts of its fanins). When each equivalence class has only one node, this computation is the same as the one presented in Section III. As before, cut enumeration is done in one topological pass from the PIs to the POs.

Example: Consider the computation of the three-feasible cuts of the equivalence class $\{o\}$ in Fig. 5. Let X represent the equivalence class $\{x_1, x_2\}$. Now, $\Phi(X) = \Phi(x_1) \cup \Phi(x_2) = \{\{x_1\}, \{x_2\}, \{q, r\}, \{p, s\}, \{q, p, e\}, \{p, d, r\}, \{p, d, e\}, \{b, c, s\}\}$. We have $\Phi(\{o\}) = \Phi(o) = \{\{o\}\} \cup \{u \cup v | u \in \Phi(\{a\}), v \in \Phi(\{x_1\}), |u \cup v| \leq 3\}$. Since $\Phi(\{a\}) = \Phi(a) = \{a\}$ and $\Phi(\{x_1\}) = \Phi(X)$, we get $\Phi(\{o\}) = \{\{o\}, \{a, x_1\}, \{a, x_2\}, \{a, q, r\}, \{a, p, s\}\}$. Observe that the set of cuts of o

TABLE I
PERFORMANCE OF IMPROVED K-FEASIBLE CUT COMPUTATION (SEE SECTION VII-A)

		$K = 4$		$K = 5$		$K = 6$		$K = 7$		$K = 8$		
Name	N	C/N	T, s	C/N	T, s	C/N	T, s	C/N	T, s	C/N	T, s	L/N, %
alu4	2642	6.7	0.00	12.3	0.01	23.1	0.04	45.5	0.18	94.7	1.02	0.00
apex2	2940	7.2	0.01	14.2	0.02	29.2	0.07	62.6	0.32	139.7	1.90	0.00
apex4	2017	8.5	0.00	19.5	0.03	47.0	0.10	116.3	0.62	293.5	4.49	0.10
bigkey	3080	6.6	0.01	12.1	0.02	24.2	0.05	50.1	0.20	99.7	0.84	0.00
clma	11869	8.1	0.04	18.2	0.11	44.4	0.51	114.9	3.01	306.3	20.99	1.64
des	3020	8.0	0.01	17.0	0.03	38.7	0.12	92.0	0.69	218.0	4.80	4.37
diffeq	2566	6.5	0.01	12.3	0.01	26.6	0.07	65.0	0.50	155.9	2.80	3.66
dsip	2521	6.2	0.01	10.7	0.01	20.7	0.03	42.0	0.10	86.7	0.44	0.00
elliptic	5502	6.4	0.01	10.6	0.03	18.5	0.07	36.9	0.33	83.4	2.12	0.20
ex1010	7652	9.2	0.02	23.3	0.11	61.8	0.61	165.8	4.01	438.2	30.43	1.99
ex5p	1719	9.4	0.01	24.1	0.02	66.2	0.17	188.2	1.30	514.8	10.50	14.14
frisc	5905	7.1	0.01	14.4	0.04	32.3	0.16	79.8	0.88	209.0	6.30	1.24
misex3	2441	7.7	0.01	15.7	0.02	33.3	0.08	73.7	0.38	170.7	2.48	0.00
pdc	7527	9.4	0.03	24.8	0.12	67.4	0.68	183.7	4.41	489.4	31.71	4.40
s298	2514	7.9	0.00	17.5	0.02	44.0	0.13	121.9	0.94	346.5	7.10	7.56
s38417	12867	6.6	0.03	13.5	0.10	32.0	0.46	83.1	3.24	225.9	23.72	3.38
s38584.1	11074	6.1	0.03	11.4	0.06	22.4	0.20	46.7	0.98	101.5	5.81	0.86
seq	2761	7.5	0.00	15.2	0.02	31.7	0.08	68.6	0.37	153.3	2.25	0.04
spla	6556	9.6	0.03	25.8	0.11	73.9	0.69	215.5	4.98	561.4	31.14	13.83
tseng	1920	6.5	0.01	11.8	0.01	23.5	0.04	50.6	0.21	112.7	1.32	1.35
Average	4954.65	7.56	0.01	16.22	0.05	38.05	0.22	95.15	1.38	240.07	9.61	2.94

involves nodes from the two choices x_1 and x_2, i.e., o may be implemented using either of the two structures.

The subsequent steps of the mapping process (computing delay-optimum mapping and performing area recovery) remain unchanged, except that now the additional cuts can be used for mapping at each node.

C. *Related Work*

Technology mapping for a network that encodes different decompositions originated in the context of standard cell mapping with the work of Lehman *et al.* [16]. Chen and Cong adapted this method for FPGAs in their work on SLDMap [6]; in particular, they identified large (five- to eight-input) AND gates in the subject graph and added choices corresponding to the different decompositions of the large AND gates into two-input AND gates. They used BDDs to find globally equivalent points, which limited the scalability of their approach.

This paper is an extension to FPGA mapping of our paper on standard cells [5]. It differs from SLDMap [6] in two ways. First, the use of structural equivalence checking, instead of BDDs, makes the choice detection scalable and robust. Second, instead of adding a dense set of algebraic choices by brute force, we add a sparse set of (possibly Boolean) choices obtained from synthesis. The expectation is that most of the choices added by the exhaustive algebraic decompositions only increase the runtime without being useful. In contrast, the choices added from synthesis are expected to be better since they are a result of optimization. This is supported by our experiments on standard cells [5], and we expect similar results to hold for FPGAs.

VII. EXPERIMENTAL RESULTS

The improvements in FPGA technology mapping are currently implemented in ABC [1] as command *fpga*. Cut enumeration is implemented as command *cut*.

A. *Improved Cut Computation (Runtime)*

Table I shows the results of exhaustive cut computation for the largest MCNC benchmarks. To derive AIGs used in this experiment, the benchmarks were structurally hashed and balanced first using command *balance* in ABC.

Exhaustive cut enumeration was performed for computing K-feasible cuts for $4 \leq K \leq 8$. Column N gives the number of AND nodes in the AIG for each benchmark. Columns C/N give the average number of cuts per node. Columns T give the runtime in seconds on an IBM ThinkPad laptop with 1.6-GHz CPU and 1 GB of RAM. The final column L/N lists the percentage of nodes for which the number of eight-input cuts exceeded the predefined limit (1000/node for these benchmarks). In computing cuts for $4 \leq K \leq 7$, the number of cuts per node never exceeded the limit, and, as a result, the cuts are computed exhaustively.

In summary, although the number of cuts and their computation time are exponential in the number of cut inputs (K), with the proposed improvements, all the cuts up to seven inputs can often be computed in reasonable time due to efficient cut filtering based on dominance.

B. *Improved Cut Computation (Memory)*

The second experiment presented memory requirements for the cut representation by showing the reduction in peak memory with and without cut dropping (Table II). The amount of memory used for a K-feasible cut in the ABC data structure is $(12 + 4^*K)$ bytes.

Columns labeled "Total" list the memory usage (in megabytes) for all the nondominated K-feasible cuts at all nodes. Columns labeled "Drop" list the peak memory usage (in megabytes) for the cuts at any moment in the process of cut enumeration, when the nodes are visited in topological order, and the cuts at a node are dropped as soon as the cuts at all the fanouts are computed.

TABLE II
PEAK MEMORY REQUIREMENTS, IN MEGABYTES, FOR CUTS WITH AND WITHOUT DROPPING (SEE SECTION VII-B)

	$K = 4$		$K = 5$		$K = 6$		$K = 7$		$K = 8$	
Name	Total	Drop	Total	Drop	Total	Drop	Total	Drop	Total	Drop
clma	2.56	0.10	6.60	0.22	18.09	0.54	52.03	1.47	152.55	4.07
ex1010	1.87	0.37	5.45	0.97	16.25	2.27	48.40	4.68	140.70	8.38
pdc	1.90	0.27	5.69	0.75	17.42	2.00	52.75	4.98	154.56	11.83
s38417	2.28	0.15	5.28	0.37	14.12	1.10	40.80	3.55	121.98	10.25
s38584.1	1.80	0.11	3.86	0.20	8.52	0.40	19.72	0.86	47.15	1.94
spla	1.68	0.21	5.15	0.59	16.63	1.65	53.88	4.34	154.44	10.04
Ratio	1.00	0.11	1.00	0.10	1.00	0.08	1.00	0.07	1.00	0.06

TABLE III
COMPARISON OF CONVENTIONAL ENUMERATION (ALL) AND CF FOR $K = 6$. RUNTIMES FOR THIS TABLE (AND TABLES IV AND V) ARE ON A 3-GHz INTEL PENTIUM 4 WITH 1 GB OF RAM. (SEE SECTION VII-C)

Name	Nodes		Number of cuts			Run-time (sec)	
	Total	%Dag	Total	Reduced	Tree	All	Factor
C1355	541	50.09	47245	25811	657	0.16	0.06
C1908	435	38.85	16546	10153	558	0.04	0.01
C2670	920	17.50	27734	14828	1583	0.07	0.02
C3540	1081	22.94	46392	29356	2064	0.10	0.05
C5315	1827	22.22	67118	35852	2677	0.13	0.07
C6288	2369	60.11	279197	209807	2818	0.82	0.53
C7552	2248	29.40	129161	78543	3251	0.28	0.16
b14	6296	23.60	397951	226401	11300	0.69	0.32
b15	8869	22.01	416804	265870	18102	0.65	0.35
clma	24387	11.20	739582	658314	27419	0.52	0.51
pj1	17675	19.11	781566	465554	32725	1.20	0.61
pj2	4055	14.38	110103	84839	5889	0.13	0.10
pj3	11178	22.45	523448	373960	22659	0.77	0.50
s15850	4127	25.66	108433	75509	6066	0.18	0.11
s35932	13711	35.89	341330	223743	16106	0.52	0.30
s38417	10820	25.00	294832	164264	16895	0.47	0.23
Ratio		27.53	1.00	0.64	0.04	1.00	0.55

In summary, dropping cuts at the internal nodes after they are computed and used reduces the memory requirements for the mapper by an order of magnitude on the largest MCNC benchmarks and by more then two orders of magnitude on large industrial benchmarks such as [13].

C. Computation of Factor Cuts

The computation of factor cuts described in Section IV is implemented in ABC [1]. Table III shows the number of complete factor cuts for $K = 6$ for a set of benchmarks. The column labeled "dag" shows the percentage of nodes that are dag nodes. On average, about 27% of the nodes are dag nodes. The number of reduced cuts is about 64% of the total number of cuts. Enumerating complete factor cuts is about two times faster than enumerating all cuts.

Table IV shows the number of all cuts, complete factor, and partial factor cuts for $K = 9$ for the same set of benchmarks. In some cases, not all cuts could be computed since there were too many. The columns labeled "Over" indicate the fraction of nodes at which the maximum limit of 2000 cuts was exceeded. When enumerating all cuts, the limit was exceeded in about 16% of the nodes on average. However, the reduced cut enumeration exceeded the limit in only 6.5% of the nodes. (The tree cut enumeration never exceeded the limit.) The number of complete factor cuts is about 68%, and the enumeration runs about 34% faster.

The columns under "PF" show the number of partial factor. It is seen from the table that the number of partial factor cuts is a small fraction of the total number of cuts (15%) and the time for enumerating these cuts is less than 10% of the time required to enumerate all cuts. During enumeration, only a small fraction of nodes (less than 0.5%) exceeded the limit of 2000 when computing dag cuts, and hence, those data are not shown in Table IV.

We note here that the multiplier (C6288) is a particularly interesting benchmark. In comparison with other benchmarks, it has many—about 60%—dag nodes. This negates the advantage of computing partial factor cuts as the factor trees are small, and the factor cut enumeration takes unusually long.

In summary, enumeration of factor cuts is feasible even for large cut sizes. Even for small K, enumerating complete factor cuts is significantly faster than enumerating all cuts.

D. Delay-Optimal Mapping With Factor Cuts

A prototype FPGA mapper using factor cuts was implemented in ABC [1]. Table V shows the delay and runtimes of the various modes of this mapper for $K = 9$. The first set of columns (under the heading "Lim = 2000") shows that

TABLE IV
COMPARISON OF CONVENTIONAL ENUMERATION (ALL), CF, AND PF FOR $K = 9$. NUMBER OF ALL NINE FEASIBLE CUTS IS AN UNDERESTIMATE. (SEE SECTION VII-C)

Name	Number of cuts							Run-time (sec)		
	All		CF			PF		All	CF	PF
	Total	Over	Reduced	Over	Tree	Dag	Leaf-dag			
C1355	433995	19.96	398226	16.82	657	103255	1457	10.77	9.86	1.25
C1908	247474	9.20	136755	2.30	562	53094	2582	4.06	1.94	0.51
C2670	363647	10.54	227604	2.61	3086	8412	11239	7.77	4.04	0.11
C3540	723405	18.69	531281	6.29	2818	29858	23391	16.37	12.3	0.27
C5315	822569	6.95	402009	0.71	2827	53299	12290	17.72	6.06	0.32
C6288	3232621	43.56	3220602	43.14	2818	2273878	6072	126.38	139.99	70.34
C7552	1632538	20.60	1100358	4.89	5214	186769	14341	38.64	24.93	1.52
b14	8937035	54.57	5734487	7.91	15607	507448	96713	182.01	144.51	2.78
b15	7498534	16.24	4484041	3.33	27812	995259	150682	148.60	74.91	11.86
clma	8870652	0.85	7688879	0.67	27419	364384	955803	60.29	50.3	4.13
pj1	12695024	18.91	7806133	3.55	63466	775824	443649	196.31	130.31	11.57
pj2	1633480	6.98	1315608	2.98	5944	48941	45664	24.29	20.13	0.16
pj3	8644521	18.00	5900954	5.47	69542	861144	291336	142.95	106.91	12.07
s15850	1323074	6.47	966661	3.46	8542	40838	52282	23.91	18.54	0.29
s35932	1623042	0.00	928175	0.00	16106	60029	54860	10.42	3.95	0.13
s38417	3678001	5.15	1587491	0.63	25547	69623	113578	61.24	19.1	0.69
Ratio	1.00	(16.04)	0.68	(6.55)	0.00	0.12	0.03	1.00	0.66	0.08

TABLE V
COMPARISON OF CONVENTIONAL MAPPING (ALL) AND CF WITH LIMITS OF 1000 AND 2000, AND PF WITH A LIMIT OF 2000 CUTS PER NODE. $K = 9$. (SEE SECTION VII-C)

Name	Lim = 2000				Lim = 1000				PF with Lim = 2000	
	Delay		Run-time (sec)		Delay		Run-time (sec)		Delay	Run-time(sec)
	All	CF	All	CF	All	CF	All	CF		
C1355	4	3	10.82	9.93	5	3	3.02	3.46	3	5.21
C1908	4	4	4.1	1.96	5	5	1.49	0.94	4	0.59
C2670	4	4	7.82	4.1	4	4	3.21	2.33	4	1.93
C3540	6	6	16.46	12.43	7	6	4.98	5.35	6	0.95
C5315	5	5	17.82	6.15	5	5	8.06	3.92	5	0.94
C6288	12	12	126.71	140.41	20	15	35.59	33.99	11	72.07
C7552	5	4	38.82	25.18	5	5	14.9	11.17	4	2.00
b14	10	10	183	147.17	12	10	43.86	63.21	10	6.99
b15	12	10	149.45	76.38	13	11	46.67	39.5	13	20.85
clma	9	9	61.13	51.38	9	9	40.28	36.76	10	8.36
pj1	8	8	197.64	132.27	8	8	61.83	62.99	9	22.30
pj2	4	4	24.48	20.39	5	4	9.16	8.81	4	0.27
pj3	6	6	143.89	108.18	8	7	48.76	49.99	7	24.37
s15850	8	7	24.06	18.86	9	8	10.54	8.33	7	2.12
s35932	2	2	10.59	4.15	2	2	7.46	4.14	2	0.21
s38417	5	4	61.64	19.73	6	5	24.97	12.79	4	2.90
Ratio	1.00	0.94	1.00	0.67	1.15	1.09	0.40	0.32	0.97	0.15

CF produces better results than enumerating all cuts and is faster. These columns directly correspond to the "All" and "CF" cut data shown in Table IV. Note that the suboptimality of enumerating all cuts is due to the fact that not all cuts could be computed for the nodes—there was an overflow of 16%. Also, by comparing the cut computation runtimes in Table IV with the overall mapping runtimes in Table V, we can see that the mapping runtime is dominated by cut computation. Expansion takes a small fraction of the total runtime, and on average, about 25% of the nodes needed to be expanded.

The second set of columns (under the heading "Lim = 1000") shows the effect of reducing the limit on the maximum number of cuts stored at a node. Although the cut computation is more than twice as fast, the delay is 15% worse when enumerating all cuts. CF continues to produce better delays and has shorter runtimes. The final set of columns (under the heading "PF with $\mathrm{Lim} = 2000$") shows the delay and runtime obtained with PF. Although one-step expansion from partial factor cuts may not generate all K-feasible cuts, the cuts that it does generate are competitive with those enumerated by the

TABLE VI
COMPARING FPGA MAPPER WITH IMPROVEMENTS WITH DAOmap [4] (SEE SECTION VII-E)

Name	DAOmap			ABC-baseline			ABC-choices			ABC-choices 5x		
	Delay	LUTs	T, s	Delay	LUTs	T, s	Delay	LUTs	T, s	Delay	LUTs	T, s
alu4	6	1065	0.5	6	984	0.16	6	971	1.36	6	889	6.58
apex2	7	1352	0.6	7	1216	0.19	7	1170	1.52	6	1046	7.27
apex4	6	931	0.7	6	899	0.18	6	890	1.11	6	852	5.76
bigkey	3	1245	0.6	3	805	0.18	3	805	1.05	3	695	7.00
clma	13	5425	5.9	13	4483	0.82	11	3695	10.54	11	2788	32.83
des	5	965	0.8	5	957	0.24	5	997	1.92	5	914	10.74
diffeq	10	817	0.6	10	830	0.17	9	785	1.37	9	761	5.48
dsip	3	686	0.5	3	694	0.17	3	694	0.93	3	693	5.64
elliptic	12	1965	2.0	12	2026	0.31	12	2085	1.81	12	2048	13.20
ex1010	7	3564	4.0	7	3151	0.59	7	2973	3.80	7	2749	21.42
ex5p	6	778	1.0	6	752	0.26	5	671	1.60	5	515	6.92
frisc	16	1999	1.9	15	2016	0.39	14	1971	2.21	13	1937	15.07
misex3	6	980	0.8	6	952	0.19	6	923	1.30	5	814	6.26
pdc	7	3222	4.6	7	2935	0.65	7	2592	5.28	7	2310	29.49
s298	13	1258	2.4	13	828	0.21	10	771	1.90	9	716	7.05
s38417	9	3815	3.8	9	4198	0.73	8	3144	7.58	7	3035	24.87
s38584	7	2987	27.0	7	3084	0.58	7	2754	6.66	6	2641	24.02
seq	6	1188	0.8	6	1099	0.22	5	1035	1.67	5	819	8.04
spla	7	2734	4.0	7	2518	0.60	7	2244	4.78	7	1922	23.17
tseng	10	706	0.6	10	759	0.17	9	725	0.86	8	725	4.23
Ratio	1.00	1.00	1.00	1.00	0.94	0.22	0.94	0.88	1.60	0.90	0.81	7.94

conventional procedure under the limit. Furthermore, PF is about six times faster than conventional enumeration.

We also experimented with PF for different values of K. For $K = 6$, we found that PF produces about 5% *worse* results than enumerating all cuts, although it runs about three times faster. For $K = 12$, we found that trying to enumerate all cuts leads to poor results since more than 40% of the nodes exceed the cut limit. PF works better, producing 50% smaller delay on average than exhaustive enumeration.

In summary, for large K (say 9 or 12), complete enumeration is not possible, and only a subset of cuts of a node can be stored and propagated in practice. Factor cuts provide a better alternative in this scenario since better cuts are generated and stored. Our experiments show that the use of factor cuts leads to better mapped results than reducing the limit on the total number of cuts stored at a node in conventional enumeration.

E. Improved Area Recovery

The "DAOmap" and "ABC-baseline" sections in Table VI compare the FPGA mapping results for five-input LUTs using DAOmap [4] and our mapper with improved area recovery implemented in ABC [1]. DAOmap was run on a four-CPU 3.00-GHz computer with 510-Mb RAM under Linux. ABC was run on a 1.6-GHz laptop with 1-Gb RAM under Windows. All benchmarks were preoptimized using *script.algebraic* in SIS followed by decomposition into two-input gates using the command *dmig* in RASP [10]. To ensure identical starting logic structures, the same preoptimized circuits originally used in [4] were used in this experiment. All the resulting netlists have been verified by a SAT-based equivalence checker [22].

Columns 2 and 5 give the number of logic levels of LUT networks after technology mapping. The values in these columns are equal in all but one case (benchmark *frisc*). This observation supports the claim that both mappers perform delay-optimum mapping for the given logic structure. The one difference may be explained by minor variations in the manipulation of the subject graph, such as AIG balancing performed by ABC.

Columns 3 and 6 show the number of LUTs after technology mapping. The difference between the results produced by the two mappers reflects the fact that different area recovery heuristics are used and, possibly, that ABC-baseline performs area recovery in a topological order, whereas DAOmap uses a reverse topological order.

Columns 4 and 7 report the runtimes in seconds. These include the time for reading a BLIF file, constructing the subject graph, and performing technology mapping with area recovery. The differences in runtimes are due to the differences in the basic data structures, improved cut enumeration, and scalability of the area recovery heuristics.

In summary, Table VI demonstrates that the mapper in ABC designed using the improved cut enumeration and the proposed heuristics for area recovery performs well on the selected benchmarks.

F. Lossless Synthesis

The "ABC-choices" section in Table VI gives mapping results for the same benchmarks when lossless synthesis is used. The alternative logic structures used for this were generated in ABC by applying script *choice* listed in the resource file *abc.rc* found in the ABC distribution. This script uses the original network and two snapshots of this network derived by applying two logic synthesis scripts in ABC, namely: 1) *resyn* and 2) *resyn2*. Both scripts are based on iterative application of AIG rewriting [20]. The three resulting networks are combined into a single choice network where functionally equivalent nodes are detected, as shown in Section VI. The mapping runtime listed in the "ABC-choices" section in Table VI includes the runtime of logic synthesis, choicing, and FPGA mapping with choices.

The "ABC-choices $5x$" section shows the results of the repeated application of mapping with choices. For this, the netlist mapped into LUTs by first mapping with choices was redecomposed into an AIG by factoring the logic functions of the LUTs, subjecting the result to the same lossless synthesis flow, and followed, as before, by mapping with choices. This process was iterated five times, which gradually transformed the logic structure to one better for FPGA mapping into five-input LUTs. The last column shows the runtime, in seconds, taken by the complete flow, including reading BLIF files, five iterations of logic synthesis, and five iterations of FPGA mapping with choices.

The quality of FPGA technology mapping (both delay and area) is substantially improved after several iterations of choicing and mapping with choices. Each iteration generates structural variations on the currently selected best mapping and allows the mapper to combine the resulting choices even better by mixing and matching different logic structures. Iterating the process tends to gradually "evolve" structures that are good for the selected LUT size independent of the structure of the original network.

We also compared our lossless synthesis with the technique in [6], which used associative choices for multi-input AND gates. The improvements due to these choices (5% in delay, 4% in area) are less than those due to the proposed lossless synthesis (6% in delay, 12% in area), compared to DAOmap, used as a baseline in Table VI. On the other hand, exhaustively adding associative decompositions greatly increases the total number of choices, leading to many more cuts. This slows down the mapper more than the relatively few choices added by the proposed lossless synthesis.

Regarding the theoretical time complexity of iterative mapping with choices, the complexity is bounded by the exponential time needed to detect choices. However, in practice, due to fast equivalence checking, the runtime is reasonable, as can be seen from the experimental results. The theoretical time complexity of choicing can also be made linear if choices are recorded during logic synthesis instead of being detected later.

In summary, the experiments demonstrate that lossless synthesis can substantially reduce delay and area of the mapped netlists both as a standalone mapping procedure and as a postprocessing step applied to an already computed FPGA mapping.

VIII. Conclusion

In this paper, we have taken the state-of-the-art techniques for LUT-based technology mapping, added three new improvements, and implemented all in a new FPGA mapper available in ABC [1]. The three improvements are: 1) reduction in runtime and memory requirements for cut enumeration; 2) improved area recovery through combined use of global-view and local-view heuristics; and 3) improved delay and area through the use of multiple circuit structures to mitigate structural bias during technology mapping.

These improvements are confirmed by experimental results using the new mapper. The improved area recovery procedure leads, on average, to a substantial improvement in runtime and a 6% smaller area, compared to DAOmap, while preserving the optimum delay when starting from the same logic structure. Using multiple logic structures via lossless synthesis leads to a 6% improvement in delay along with a 12% reduction in area while the runtime is slightly increased compared with DAOmap. When lossless synthesis and FPGA mapping are iterated five times, delay and area improve by 10% and 19%, respectively, at the cost of increasing the runtime eight times (which includes the extra logic synthesis time).

We also introduced the notion of cut factorization to enable delay-oriented mapping for large LUT sizes. Cut factorization can be seen as an alternative to storing a limited number of cuts at a node in conventional enumeration, and the experimental results show that using factor cut-based mapping leads to better delays and shorter runtimes than conventional enumeration.

Future Work: Confirmation of the full usefulness of factor cuts remains for future experiments. Our next goal is to apply factor cut computation for technology mapping into macrocells or configurable logic blocks in FPGAs with eight or more inputs. Macrocells differ from LUTs in that they can implement a subset of all functions of the given number of inputs. Another possibility is to use factor cuts in standard cell mapping and in logic synthesis by rewriting [20]. The cut size correlates with the capability of a mapper (or a rewriting algorithm) to overcome structural bias. The larger is the cut, the larger is the scope of Boolean matching (or Boolean transform), and the smaller is the structural bias.

Also, a major work for the future will be to extend the improvements to FPGA mapping for the case of integrated sequential optimization, which includes logic restructuring, mapping, and retiming, as presented in [21].

Acknowledgment

The authors would like to thank J. Cong and D. Chen for providing the set of preoptimized benchmarks from [4], which allowed for a comparison with DAOmap in Table VI. The authors also acknowledge N. Een for suggesting the counter-example shown in Fig. 3.

References

[1] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/

[2] V. Bertacco and M. Damiani, "Disjunctive decomposition of logic functions," in *Proc. ICCAD*, 1997, pp. 78–82.

[3] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," in *Commun. ACM*, 1970, vol. 13, pp. 422–426.

[4] D. Chen and J. Cong, "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," in *Proc. ICCAD*, 2004, pp. 752–757.

[5] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," in *Proc. ICCAD*, 2005, pp. 519–526. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/publications/2005/iccad05_map.pdf

[6] G. Chen and J. Cong, "Simultaneous logic decomposition with technology mapping in FPGA designs," in *Proc. FPGA*, 2001, pp. 48–55.

[7] J. Cong and Y. Ding, "FlowMap: 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. 1–12, Jan. 1994.

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

[9] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proc. FPGA*, 1999, pp. 29–36.

[10] J. Cong *et al., RASP: FPGA/CPLD Technology Mapping and Synthesis Package*. [Online]. Available: http://ballade.cs.ucla.edu/software_release/rasp/htdocs/

[11] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Proc. SAT*, 2005, pp. 61–75.

[12] A. Farrahi and M. Sarrafzadeh, "Complexity of 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.

[13] IWLS 2005 Benchmarks. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[14] C.-C. Kao and Y.-T. Lai, "An efficient algorithm for finding minimum-area FPGA technology mapping," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 168–186, Jan. 2005.

[15] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1377–1394, Dec. 2002.

[16] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 8, pp. 813–833, Aug. 1997.

[17] F. Lu, L. Wang, K. Cheng, J. Moondanos, and Z. Hanna, "A signal correlation guided ATPG solver and its applications for solving difficult industrial cases," in *Proc. DAC*, 2003, pp. 668–673.

[18] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," in *Proc. IWLS*, 2004, pp. 14–21.

[19] A. Mishchenko, S. Chatterjee, R. Jiang, and R. Brayton. (2005, Mar.). "FRAIGs: A unifying representation for logic synthesis and verification," EECS Dept., UC Berkeley, Berkeley, CA, ERL Tech. Rep. [Online]. Available: http//:www.eecs.berkeley.edu/~alanmi/publications/2005/tech05_fraigs.pdf

[20] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *Proc. DAC*, 2006, pp. 332–536. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/publications/2006/dac06_rwr.pdf

[21] A. Mishchenko, S. Chatterjee, R. Brayton, and P. Pan, "Integrating logic synthesis, technology mapping, and retiming," UC Berkeley, Berkeley, CA, Apr. 2006. ERL Tech. Rep. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/publications/2006/tech06_int.pdf

[22] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Eén, "Improvements to combinational equivalence checking," in *Proc. ICCAD'06*, 2006, pp. 836–843. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/publications/2006/iccad06_cec.pdf

[23] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for LUT-based FPGAs," in *Proc. FPGA*, 1998, pp. 35–42.

Alan Mishchenko (M'99) received the M.S. degree in applied mathematics and information technology from the Moscow Institute of Physics and Technology, Moscow, Russia, in 1993, and the Ph.D. degree in computer science from the Glushkov Institute of Cybernetics, Kiev, Ukraine, in 1997.

Since 1998, he has been a Research Scientist in the U.S. First, he was with the Portland State University, Portland, OR. Since 2002, he has been with the University of California, Berkeley. His research focuses on developing computationally efficient methods for logic synthesis and verification.

Satrajit Chatterjee received the B.Tech. and M.Tech. degrees from the Indian Institute of Technology, Bombay, India, in 2001. He is currently working toward the Ph.D. degree in computer science at the University of California, Berkeley.

His research interests are in developing algorithms for logic synthesis and physical design.

Robert K. Brayton (M'75–SM'78–F'81) received the B.S. degree in electrical engineering from Iowa State University, Ames, in 1956, and the Ph.D. degree in mathematics from Massachusetts Institute of Technology (MIT), Cambridge, in 1961.

From 1961 to 1987, he was with the Mathematical Sciences Department, IBM T. J. Watson Research Center, Yorktown Heights, NY. In 1987, he joined the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, where he is currently a Cadence Distinguished Professor of engineering and the Director of the Semiconductor Research Corporation Center of Excellence for Design Sciences.

Dr. Brayton is a member of the National Academy of Engineering and a Fellow of the American Association for the Advancement of Science. He received the 1991 IEEE Circuits and Systems Technical Achievement Award and five best paper awards, including the 1971 IEEE Guilleman–Cauer Award and the 1987 International Symposium on Circuits and Systems Darlington Award. In 2000, he received the Circuits and Systems Golden Jubilee Medal and the IEEE Millennium Medal, the 2002 Iowa State Marston Medal, and the 2006 IEEE Piore Award.