# Simultaneous Logic Decomposition with Technology Mapping in FPGA Designs

Gang Chen and Jason Cong
Computer Science Department
University of California, Los Angeles, CA 90095
{chg, cong}@cs.ucla.edu

## ABSTRACT

Conventional technology mapping algorithms for SRAM-based Field Programmable Gate Arrays (FPGAs) are normally carried out on a fixed logic decomposition of a circuit. The impact of logic decomposition on delay and area of the technology mapping solutions is not well understood. In this paper, we present an algorithm named *SLDMap* that performs delay-minimized technology mapping on *a large set* of decompositions and simultaneously controls the mapping area under delay constraints. Our study leads to two conclusions: (1) For depth minimization, the best algorithms in conventional flow (*dmig* + *CutMap*) produce satisfactory results with a short runtime, even with a fixed decomposition; (2) When all the structural decompositions of the 6-bounded Boolean network are explored, *SLDMap* consistently outperforms the state-of-the-art separate flow (*dmig* + *CutMap*) by 12% in depth and 10% in area on average; it also consistently outperforms the state-of-the-art combined approach *dogma* by 8% in depth and 6% in area on average.

## 1. Introduction

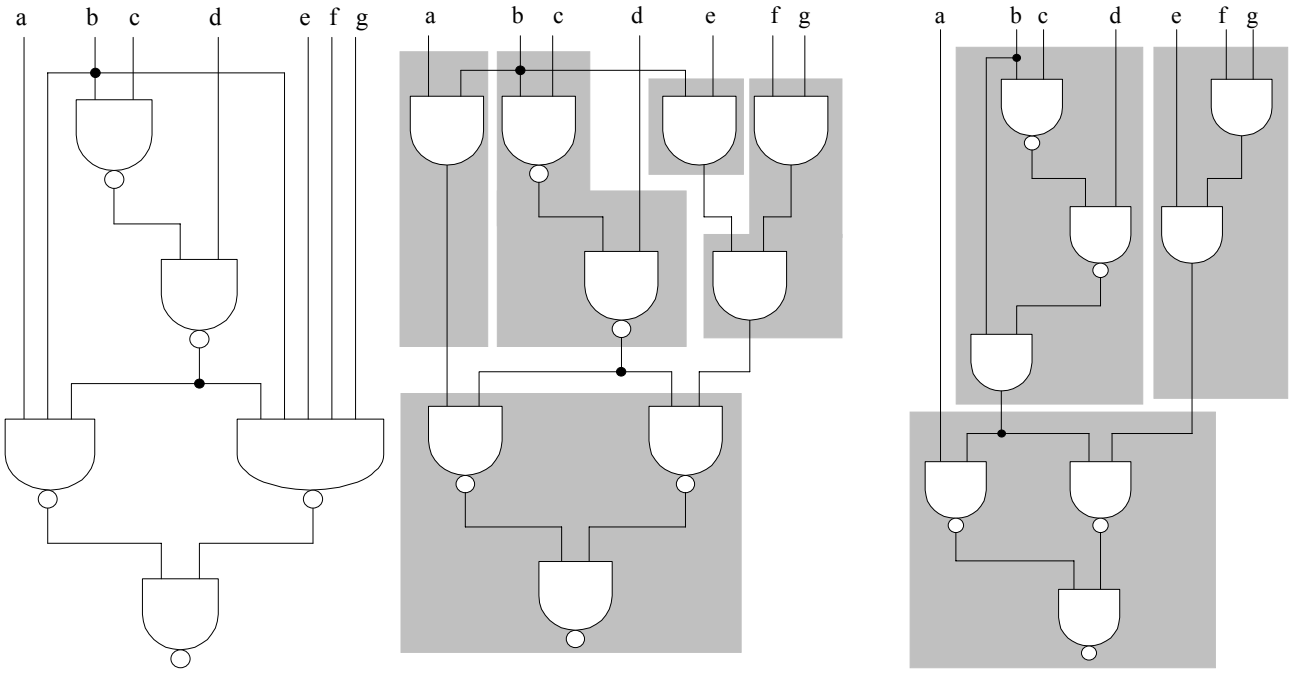
Field Programmable Gate Arrays (FPGAs) have become more and more popular in recent years because of their short time-to-market, field programmability, ease of use, and low cost in small- to medium-volume production. A typical type of FPGA is based on a K-input lookup-table (LUT) kind of cell, which can implement arbitrary K-input functions. Technology mapping is an essential step in FPGA synthesis, and there have been very extensive studies on the problem. However, in most conventional approaches, mapping is applied on a fixed logical decomposition of the circuit.

The conventional flow for FPGA mapping consists of three phases. During the first phase, technology-independent optimizations are applied on the initial circuit. Both Boolean and algebraic methods are used, such as: kernel/cube extraction, node substitution, don't care-based optimizations, etc. Then, during the logic decomposition phase, large gates are decomposed into K-bounded gates (gates with less than K inputs). After that, technology mapping is applied on the K-bounded network. The drawback for the separate approach is that during the technology-independent optimization and decomposition stage, we have great freedom to go from one solution to another, but a fast and accurate estimation of the final mapping result is not available. During the technology mapping stage, we are able to perform depth-optimal mapping, but the solution space is greatly confined because we are committed to a fixed circuit structure as the starting point for mapping. Since the delay-optimal decomposition for mapping is NP-hard [1], it is unclear how far away we are from the optimal solution when logic decomposition and technology mapping are performed simultaneously. The goal of this research is to combine logic decomposition and technology mapping for LUT-based FPGA designs, with delay minimization as the primary objective.


*FPGA 2001,* February 11-13, 2001, Monterey, California, USA.


The delay of a LUT network can be roughly measured by the depth of a network. We first review the existing algorithms on FPGA mapping for delay optimizations. *Chortle-d* [2], *MIS-pgad* [3] and *DAGMap* [4] are early attempts to minimize circuit delays. *Chortle-d* decomposes the network into fanout-free trees, maps each tree optimally using dynamic programming and bin-packing techniques, then combines the solutions for each tree. It results in sub-optimal depth and uses a larger area. *MIS-pgad* makes use of dynamic resynthesis techniques and layout information during mapping, but overall it produces larger depth than *Chortle-d*. The *dmig* [4] (decompose-multi-input-gate) algorithm transforms an arbitrary simple-gate network into a two-input network, and guarantees that the transformed network has the smallest possible depth. *DAGMap* first applies the *dmig* algorithm to generate a depth-optimal 2-bounded network, then it generates the mapping solution on the general network directly using *the Lawler labeling*, and finally *gate decomposition* and *predecessor packing* are used to minimize the area as post-processing. *DAGMap* produces both better delay and area result than *Chortle-d* and *MIS-pgad*, but it is depth-optimal only for trees. *FlowMap* [5] is the first depth-optimal polynomial time algorithm for general K-bounded Boolean networks. It uses flow computation to label each node with its minimum possible depth. Compared to *DAGMap*, *FlowMap* achieves a small delay reduction but a sizeable area reduction. Afterwards, *FlowMap-r* [6] is able to trade the depths of nodes on non-critical paths or even the depth of the entire network for a smaller area. *CutMap* [7] performs simultaneous delay and area minimization. It is able to outperform *FlowMap* in area by 15% while maintaining the optimal depth. *CutMap* is used in this work for comparison. The general FPGA mapping problem for area minimization is NP-hard [8]. A more extensive survey of LUT-based technology mapping is available in [9].

Previous studies on simultaneous decomposition and mapping went back to *Chortle-d*, which guarantees depth-optimal technology mapping for simple-gate *tree* networks. Afterwards, *dogma* [1] studied structural gate decomposition for depth-optimal technology mapping of general networks. Although the problem of delay-optimal *structural decomposition* is NP-hard, the work shows that




(a) Initial 5-bounded network

(b) Result from dmig decomposition and the best mapping on it with depth of 3 and area of 5

(c) Result from optimal simultaneous decomposition with mapping with depth of 2 and area of 3

**Figure 1. Impact of gate decomposition on technology mapping**

*structural decomposition* will result in better final implementation in terms of both delay and area over previous separate decomposition methods (*tech_decomp*, *dmig* followed by *FlowMap* or *Chortle-d*). In parallel, Lehman et al. developed a novel method [10][11] of performing logic decomposition during technology mapping for library-based designs. They showed that there is a lot of room in delay reduction for library-based mapping if decomposition and mapping are combined in one step. Recently, [12] shows that area-delay trade-offs can also be obtained using the combined approach.

For FPGA mapping, logic decompositions could significantly affect the final mapping depth and area. For example, given a 5-bounded Boolean network in Figure 1(a), assume that the 3-LUT is used for the mapping. A mapping solution is a network of LUTs. The mapping depth is defined to be the number of levels of the LUT network, and the mapping area is defined to be the number of LUTs in the network. The conventional state-of-the-art gate decomposition algorithm (*dmig*) will generate a 2-bounded network of four levels, as shown in Figure 1(b). The best result we can obtain from *dmig* decomposition is depth 3 and area 5. However, a mapping solution of depth 2 and area 3 can be obtained from the gate decomposition of Figure 1(c).

The objective of this research is to evaluate the impact of the logic decomposition on both *delay* and *area* of the final mapping solutions. We perform logical decomposition and technology mapping simultaneously and explore the entire solution space of *all structural decompositions* for a W-bounded Boolean network. Compared with the conventional approach (*dmig* + *CutMap*), our approach reduces the LUT network depth by up to 40% (12% on average) and reduces the number of LUTs by up to 57% (10% on average).

# 2. Problem Formulation and Preliminaries

## 2.1 Definitions

We consider FPGA synthesis for combinational circuits in this work. A combinational Boolean network $N$ can be represented by a directed acyclic graph (DAG) $N=(V, E)$, in which each node $v \in V$ represents a logic gate and each directed edge $(u, v)$ represents a wire connecting the output of gate $u$ to one input of gate $v$. A primary input (PI) is a node without incoming edges; a primary output (PO) is a node without outgoing edges. A node $v$ is K-bounded if and only if $|fanin(v)| \leq K$; a network $N$ is K-bounded if and only if all the nodes in $N$ are K-bounded. The depth (or level) of a node $v$ is the number of edges on the longest path from any PI to $v$. Obviously all PIs have a depth of 0. The depth of a network is the maximum depth of all nodes in the network. A mapping solution of a Boolean network $N$ is a DAG in which each node is a K-LUT and the edge $(C_w, C_v)$ exists if the output of $C_u$ is connected to one input of $C_v$. The mapping depth of the LUT network is the maximum mapping depth of all POs; the mapping area is the number of LUTs in the mapped network. The *minimum mapping depth* of $N$, denoted as *MMD*, is the minimum network depth of all mapping solutions of $N$. The label of a node is the minimum mapping depth of the node. Let *fanin(v)* and *fanout(v)* represent the set of fanins and the set of fanouts of node $v$, respectively. Given a subgraph $H$ of a network $N$, let *fanin(H)* denote the set of distinct nodes outside $H$ that supply inputs to nodes inside $H$. A node $u$ is a predecessor of a node $v$ if there exists a directed path from $u$ to $v$ in network $N$. A fanin cone $C_v$ rooted at $v$ is a connected sub-network, which consists of only $v$ and its predecessors. All the directed paths connecting any two nodes $u_1$ and $u_2$ in $C_v$, lie entirely in $C_v$. A cut is a partitioning $(X, X')$ of a cone $C_v$ such that $X'$ is a cone of $v$. The node cut-set of the cut, denoted $V(X, X')$, consists of the inputs of cone $X'$. A cut is K-

feasible if $|V(X, X')| \leq K$. The height of a cut $(X, X')$, denoted $h(X, X')$, is defined to be the maximum label in $X$, i.e., $h(X, X') = max \{label(x) | x \in X\}$.

Gate decomposition methods include *structural*, *algebraic*, *Boolean* and *functional* approaches. Simple gates consist of AND, OR, XOR gates or their inversions, and a simple-gate network is a network consisting of simple gates only. *Structural gate decompositions* can only be applied to simple gates. The *tech_decomp* algorithm in SIS [13], the *dmig* algorithm, and the *Chortle* mapping algorithm family all carry out structural gate decomposition. The *dogma* algorithm performs structural decomposition and technology mapping simultaneously and gets better mapping solutions than *dmig* in both delay and area. In *algebraic decomposition* approaches, networks are usually partially collapsed, and gates are represented in the sum-of-product (SOP) form. Common logic sub-functions are then extracted with algebraic divisions. In *Boolean decomposition* approaches, logic gates are decomposed via functional operations. Shannon expansion, if-then-else (ITE) decomposition, and AND-OR decomposition are very common Boolean decomposition operations. In *functional decomposition* approaches, networks are completely collapsed whenever possible for the outputs to be directly represented as functions of the network inputs. The output functions are then decomposed into composed K-input sub-functions for implementation using K-LUTs. Optional LUT mapping steps may follow to improve the synthesis results. In general, algebraic and Boolean approaches are more effective for both area and delay minimization while *structural* approaches are usually faster.

The problem we are going to solve in this work is **structural gate decomposition in a W-bounded network for K-LUT mapping (W-SGD/K)**: given a simple-gate W-bounded network $N_W$, decompose $N_W$ into a 2-bounded network $N_2$ such that for any other 2-bounded decomposed network $N'_2$, $MMD(N_2) \leq MMD(N'_2)$.

## 2.2 Complexity Results

We would like to briefly review some existing complexity results on structural decomposition for FPGA mapping in [1].

**Theorem 1 [1]**
Given a W-bounded network $N$, if only structural gate decomposition is allowed, the minimum mapping depth for all integrated mapping approaches equals the minimum mapping depth for all separate mapping approaches (i.e., generate a fixed decomposition followed by technology mapping).

**Theorem 2 [1]**
The W-SGD/K problem is NP-hard for $W = K \geq 5$.

**Corollary 2.1**
The W-SGD/K problem is NP-hard for $W \geq K \geq 5$.

**Proof:** If W-SGD/K is not NP-hard for $W > K \geq 5$, then any W bounded simple-gate network can be decomposed into a speed-optimal 2-bounded network in polynomial time when $W > K \geq 5$. For any K bounded network, it is also W bounded (when $W > K \geq 5$), so it can also be decomposed into a speed-optimal 2-bounded network in polynomial time. This implies W-SGD/K is not NP-hard for $W = K \geq 5$, which is contradictory to Theorem 2.

# 3. Description of *SLDMap*

## 3.1 Review of [11]

In [11], Lehman et al. propose an algorithm that performs logic decomposition and technology mapping simultaneously for library-based designs. The mapping procedure effectively explores all decompositions encoded in a mapping graph and generates a delay optimal tree implementation. One problem with this algorithm is that area is uncontrolled and sometimes becomes unnecessarily large. Here are some preliminaries from their work.

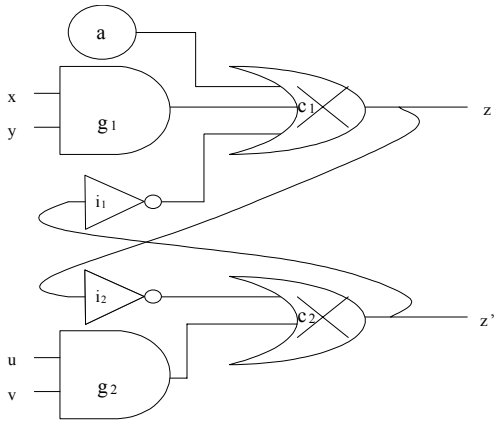



**Figure 2. Illustration of mapping graph**

A Boolean network is called an AND2/INV network when every internal node is either an inverter (INV) or a 2-input AND (AND2). The *AND2/INV decomposition* decomposes a simple-gate network into an AND2/INV network. A mapping graph is a data structure based on an AND2/INV Boolean network with four modifications. First, a *choice node*, whose fanin nodes are all logically equivalent, is introduced. Second, directed cycles may exist in a mapping graph. Third, a *ugate* is a collection of nodes in a mapping graph consisting of two complementary choice nodes and all of their immediate fanins. Fourth, each choice node in a reduced mapping graph is unique, i.e., any two choice nodes represent different logic functions. Figure 2 is an illustration of the mapping graph. $c_1$ and $c_2$ are two complimentary choice nodes, $a$ is a primary input, $g_1$ and $g_2$ are two 2-input AND gates, and $i_1$ and $i_2$ are two inverters. From this mapping graph, one can tell that the primary input $a$ is logically equivalent to $(x \wedge y)$ and the inversion of $(u \wedge v)$, gates $c_1$, $i_2$, $c_2$ and $i_1$ form a cycle, and all these nodes together form a ugate.

For Boolean networks, there are three types of transformations: *associative*, *distributive* and *inverter transformations* (Figure 3). The *associative transformation* is based on the associative law: $(xy)z = x(yz)$. The *distributive transformation* is based on the distributive law: $(xy + xz)' = (x(y + z))'$. The *inverter transformation* adds or removes two consecutive inverters between two consecutive nodes in a mapping graph. Let *replacement transformation* model the effect of the reduction operation on a mapping graph, which joins two or more logically equivalent mapping graphs into one. For a single AND2/INV decomposition $\alpha$ of network $N$, if *associative*, *inverter*, and *replacement transformations* are exhaustively applied on $\alpha$, the resulting mapping graph is denoted as $\Lambda_N$.

**Theorem 3 [11]**
Every AND2/INV decomposition of a Boolean network $N$ is contained in $\Lambda_N$.

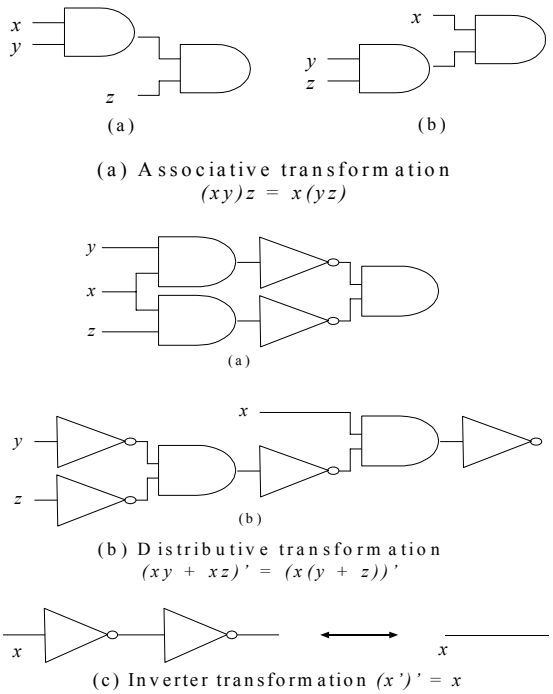



**Figure 3. Three kinds of logic transformations**

## 3.2 Overview of *SLDMap*

This work extends the basic framework of [11] from library-based mapping to FPGA mapping, but it also successfully integrates into the framework the labeling technique and cut enumeration technique which have proven successful in FPGA mapping. The results in [11] show that a remarkable delay reduction in library cell mapping can be obtained if logic decomposition and technology mapping stages are combined. Therefore, in this work we want to evaluate such an impact on FPGA mapping. The problem itself is very difficult for FPGAs, since the depth-optimal logic decomposition should be identified with accurate area estimations at the same time for simultaneous area minimization. Both the W-SGD/K [1] and area minimization LUT mapping [8] problems are NP-hard. The following is the outline of our approach:

(1) An arbitrary initial decomposition is obtained by using either *tech_decomp*, *dmig* or *dogma*.

(2) A new W-bounded network is generated by collapsing consecutive AND gates into larger W-bounded AND gates. The mapping graph is then constructed based on the new W-bounded network to encode *all possible decompositions* for each gate.

(3) A depth minimization mapping is performed on the mapping graph to label each ugate with the smallest possible depth with consideration of all possible decompositions.

(4) Under the minimized depth constraint, label relaxation is performed to get better area without affecting the network depth.

(5) The best decomposition is selected from all decompositions encoded in the mapping graph.

(6) The state-of-the-art mapping algorithms for a fixed decomposition (*CutMap and Greedy_Pack)* are performed on the selected decomposition to obtain the final mapping solution.

The following sections discuss steps (2) to (5) in more detail.

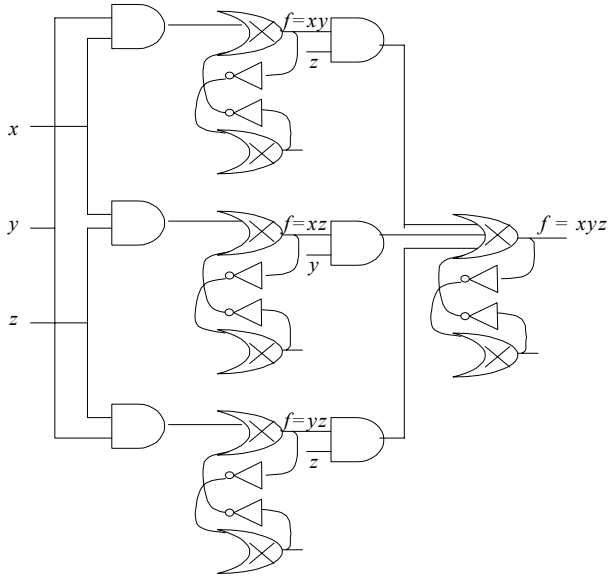



**Figure 4. A mapping graph which encodes all decompositions of *f = xyz***

## 3.3 Mapping Graph Construction

The construction of a mapping graph consists of two phases.

First, a given decomposition is mapped into an AND2/INV network. This is simply done by using the "map" command in SIS [13] to map the decomposition into a library consisting of AND2 and inverter only. This is a preprocessing step, and its optimality is not important. Then consecutive AND2s are collapsed into bigger W-bounded AND gates.

Second, all nodes are processed one by one in topological order. If a node is a PI or Inverter, a corresponding ugate will be created; if it is an AND, all possible *structural decompositions* of the AND gate will be enumerated and connected to a choice node. One example of a mapping graph is given in Figure 4, where all possible decompositions of $f = (xy)z = (xz)y = (yz)x$ are succinctly represented. For a W-input AND gate, the number of different decompositions is $((2W)!/W!2^W(2W-1))$ [11], and the number of ugates introduced is $2^W-W-1$ [12]. Note that the binary decision diagram (BDD) is used during the mapping graph construction to identify functionally equivalent nodes for graph reduction. However, when W is not too large, both numbers can be regarded as constants. For example, when $W = 6$, the number of different decompositions is 945, and the number of ugates introduced is 57 for one 6-input AND gate. So both the complexity of the mapping graph construction algorithm and the increase in the mapping graph size are linear (bounded by a large constant).

## 3.4 Mapping for Depth Minimization

Before the detailed description of our mapping algorithm, we first take a look at the complexity of the problem itself. During the phase of mapping graph construction, all neighboring AND2s are collapsed into W-bounded AND gates, then all the structural

decomposition of theses AND gates are enumerated in the mapping graph. If we could obtain the delay-optimal decomposition in the mapping graph, then we actually could solve the *W-SGD/K* problem. As we know in **Corollary 2.1**, *W-SGD/K* is NP-hard for $W \geq K \geq 5$, so the mapping graph mapping problem is NP-hard for $W \geq K \geq 5$.

**Corollary 2.2**

For a mapping graph constructed from a W-bounded network *N* containing every structural decomposition of *N*, depth-optimal mapping is NP-hard for LUT-based FPGA mapping when $W \geq K \geq 5$.

Our delay minimization technology mapping *SLDMap* is applied directly on the mapping graph without dividing the graph into trees. *SLDMap* combines the labeling technique in *DAGMap* and the cut enumeration technique in *Preator* [14].

First, all the nodes are sorted in *pseudo topological order* starting from PI nodes, i.e., orders increase from primary inputs to primary outputs, and the same order is assigned to all ugates in a cycle. Mathematically, if there exists a directed path from ugate $u_1$ to ugate $u_2$, then $order(u_1) \leq order(u_2)$; if there exists no directed path from either $u_1$ to $u_2$ or $u_2$ to $u_1$, then $order(u_1) \neq order(u_2)$; if there exists both directed paths from $u_1$ to $u_2$ and from $u_2$ to $u_1$, then $order(u_1) = order(u_2)$.

Second, each ugate u with PI fanin is labeled $h(u) = 0$, and the cut-set of *u* is *u* itself ($cut\text{-}set(u) = \{u\}$). For large circuits that need to be partitioned, pseudo PIs and POs should be created across the cut-line, and each pseudo PI should be labeled with the label of the pseudo PO it is connected to.

Third, each ugate *u* is labeled with its minimum depth *h(u)* in *pseudo topological order*, and all K-feasible cuts of height *h(u)* are recorded in *u*. Each branch of fanin to a ugate *u* represents one possible implementation, and the minimum label of all branches should be used to label *h(u)*. If a branch *v* is a PI node, then *label(v)* is 0. If a branch *v* is an AND2, let $u_1$ and $u_2$ be the two ugates supplying inputs to the AND2 gate *v*, and *p* be $max(h(u_1), h(u_2))$. All recorded cut-sets of $u_1$ and $u_2$ are combined to form new cut-sets. Of all the new cut-sets generated, if there exists K-feasible cuts of height *p*, the *label(v)* is *p* and all such K-feasible cuts are temporarily stored at AND2 gate *v*; otherwise *label(v)* is $p + 1$ and $\{u_1, u_2\}$ are stored. After all the branches are labeled, the minimum label of them $(\min_{v \in fanin(u)} label(v))$ is selected for *h(u)*, and all K-feasible cuts of *h(u)* are kept at *u*. For ugates in a cycle, the labeling process needs to be repeated within the cycle until the label of each ugate remains unchanged. In our experiment, the labels always become stable in less than five iterations.

Figure 5 is a portion of the mapping graph that encodes both the *dmig* decomposition in Figure 1(b) and the optimal decomposition in Figure 1(c). Assume K is 3 for the LUT mapping. The pair *(1, {b,c})* for ugate $u_1$ means $u_1$'s minimum depth *h(u)* is 1, its cut-set is *{b, c}*. Ugate $u_9$ has two branches that both generate a depth of 2, so both cut-sets $\{u_1, u5\}$ and $\{a, u_8\}$ are kept; ugate $u_{10}$ has two

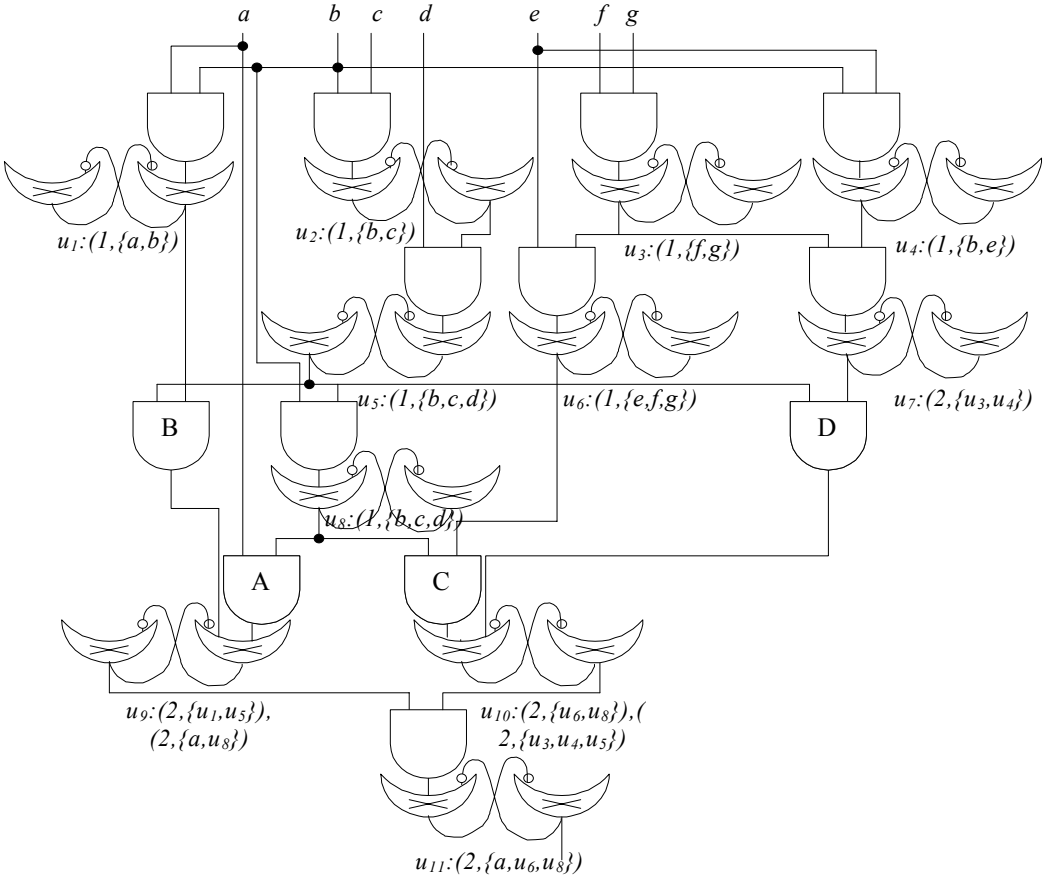



**Figure 5. Labeling of ugates in the mapping graph of network in Figure 1**

branches which both generate a depth of 2, so both cut-sets $\{u_6, u_8\}$ and $\{u_3, u_4, u_5\}$ are kept. When cut-sets at $u_9$ and $u_{10}$ are combined, the only 3-feasible cut-set of height 2 is $\{a, u_6, u_8\}$, so ugate $u_{11}$'s depth is labeled 2.

**Theorem 4**
The mapping graph LUT mapping algorithm is at least as good as performing *DAGMap* on *all the structural decompositions* of the original W-bounded network.

**Proof:** Please refer to [15].

In practice, the procedure is not very time-consuming since most combinations of cut-sets of $u_1$ and $u_2$ result in K-infeasible cut-sets and hence cannot be further propagated. Also, if u is assigned to a depth of $max(h(u_1), h(u_2)) + 1$, the only cut-set kept is $\{u_1, u_2\}$. In our final implementation, we decided to keep only one cut-set with minimum cut-size among all K-feasible cut-sets for each ugate. Experimental results show that there are no observable differences in depth between this heuristic and the optimal *DAGMap* mapping for the mapping graph. When only one cut is kept for each ugate, the mapping algorithm's complexity is linear to the size of the mapping graph.

## 3.5 Area Relaxation

A reverse pass through the mapping graph can be performed by working backward recursively from each primary output and fixing the decomposition in the depth minimization mapping stage. The drawback for this simple approach is that the fastest solutions on *all* paths instead of *critical* paths are kept. This naturally consumes a bigger area than necessary. Off the critical paths, solutions just fast enough and with the smallest possible area consumption should be preferred.

To obtain the optimal solution, both delay and area should be stored at each cut of a ugate. For standard cell mapping, only the Pareto points need to be stored to get the optimal solution [12]. But for FPGA mapping, this is not the case. In our approach, the first five smallest solutions with the same depth are kept for each ugate, and propagated from PIs to POs. Then from each PO ugate, the solution with fast enough speed and smallest possible area is selected, and a reverse pass is performed to select the best decomposition for the entire network encoded in the mapping graph.

In our implementation, in order to avoid further complications induced by cycles in the mapping graph for area estimation, graph mapping is performed multiple times. Whenever cycles are detected, heuristics are used to remove them. All ugates with the minimum label in a cycle are selected, and their branches of fanin AND2s with both input ugates in the same cycle are removed. The procedure is repeated until no cycles exist in the mapping graph.

During the area control phase, there are three steps: label relaxation, forward pass and backward pass.

Label relaxation adds required arrival time (RAT) to each ugate in the mapping graph. The depth of the entire network is assigned to each PO ugate's RAT, and propagated backward to all internal ugates in the reverse topological order.

The forward pass generates all the delay/area pairs and propagates from PIs to POs. At each ugate $u$, only cuts with a height smaller than or equal to $RAT(u)$ are kept. For cuts of the same depth, the first five cuts with the minimum area are kept. For area estimation, several heuristics have been tested:

(1) $area(u) = \sum_{v \in cut\text{-}set(u)} area(v) + 1$;

(2) $area(u) = \sum_{v \in cut\text{-}set(u)} area(v)/num\_fanout(v) + 1$;

(3) $area(u) = \sum_{v \in cut\text{-}set(u)} area(v)/crossing\_num(v) + 1$ [12].

The experiments show that the simplest estimation (1) gives the best mapping area in the shortest amount of time.

In the backward phase, we select the delay/area pairs with minimum area and fast enough speed from each PO and propagate all the way back to PIs. The best decomposition is selected on the fly during this phase. The resulting decomposition is an AND/INV 2-bounded network, on which the state-of-the-art mapping algorithm *CutMap* is applied to generate the final mapping solutions.

# 4. Experimental Results

The UCLA RASP package [16] and CUDD package [17] are used for the experiments. As a pre-processing step, *script.rugged* is performed on all original MCNC circuits for both area and delay minimization, and *tech_decomp –a 1000 –o 1000* is used to generate a simple-gate network. Then, *dmig*, *dogma*, and *SLDMap* are applied on the simple-gate networks for comparison. K (maximum number of inputs to a LUT) is set to 5, and W can be chosen from 2 to 9. W cannot be larger than 10, since for a W-input AND, the number of different decompositions is $((2W)!/W!2^W(2W-1))$, and the number of ugates introduced is $2^W-W-1$.

A comparison is made with the conventional separate approach (*dmig* + *DAGMap/CutMap*), and a combined approach (*dogma* [1] + *DAGMap/CutMap*). *CutMap* [7] is used instead of FlowMap because it produces the same optimal mapping depth while using fewer LUTs in general. The *Greedy_Pack* algorithm is applied on all the mapping solutions as a post-processing step to further minimize area.

**Table 1. Comparison of *SLDMAP* with *dmig*[4] and *dogma*[1] using *DAGMap*** (D: Delay; A: Area; R: Ratio to *SLDMap*)

| circuit | dmig | | | | dogma | | | | SLDMap | |
|---|---|---|---|---|---|---|---|---|---|---|
| | D | R | A | R | D | R | A | R | D | A |
| count | 5 | 1.67 | 46 | 0.71 | 5 | 1.67 | 43 | 0.66 | 3 | 65 |
| 9symml | 5 | 1.00 | 129 | 1.32 | 5 | 1.00 | 100 | 1.02 | 5 | 98 |
| frg1 | 5 | 1.25 | 67 | 0.93 | 4 | 1.00 | 82 | 1.14 | 4 | 72 |
| i3 | 4 | 1.33 | 154 | 2.33 | 4 | 1.33 | 138 | 2.09 | 3 | 66 |
| alu2 | 7 | 1.00 | 139 | 1.10 | 7 | 1.00 | 136 | 1.08 | 7 | 126 |
| x1 | 3 | 1.00 | 136 | 1.08 | 3 | 1.00 | 134 | 1.06 | 3 | 126 |
| C432 | 10 | 1.00 | 130 | 1.23 | 10 | 1.00 | 131 | 1.24 | 10 | 106 |
| alu4 | 9 | 1.00 | 258 | 0.99 | 9 | 1.00 | 275 | 1.06 | 9 | 260 |
| rot | 7 | 1.17 | 298 | 0.96 | 7 | 1.17 | 292 | 0.94 | 6 | 312 |
| i2 | 5 | 1.25 | 119 | 1.51 | 4 | 1.00 | 79 | 1.00 | 4 | 79 |
| C880 | 8 | 1.00 | 127 | 1.01 | 8 | 1.00 | 125 | 0.99 | 8 | 126 |
| C2670 | 9 | 1.13 | 271 | 0.91 | 9 | 1.13 | 268 | 0.90 | 8 | 298 |
| dalu | 5 | 1.00 | 454 | 1.38 | 5 | 1.00 | 411 | 1.25 | 5 | 329 |
| C3540 | 10 | 1.00 | 608 | 1.04 | 10 | 1.00 | 599 | 1.02 | 10 | 587 |
| toplarge | 5 | 1.00 | 176 | 1.02 | 5 | 1.00 | 196 | 1.13 | 5 | 173 |
| t481 | 6 | 1.20 | 181 | 1.06 | 6 | 1.20 | 169 | 0.99 | 5 | 171 |
| k2 | 6 | 1.00 | 469 | 1.00 | 6 | 1.00 | 482 | 1.03 | 6 | 468 |
| C7552 | 8 | 1.00 | 651 | 1.08 | 8 | 1.00 | 647 | 1.07 | 8 | 605 |
| des | 5 | 1.00 | 1316 | 1.00 | 6 | 1.20 | 1304 | 0.99 | 5 | 1312 |
| C499 | 4 | 1.00 | 66 | 0.93 | 4 | 1.00 | 72 | 1.01 | 4 | 71 |
| apex6 | 6 | 1.20 | 280 | 1.07 | 5 | 1.00 | 274 | 1.05 | 5 | 262 |
| apex7 | 4 | 1.00 | 80 | 1.08 | 4 | 1.00 | 82 | 1.11 | 4 | 74 |
| duke2 | 4 | 1.00 | 201 | 1.00 | 4 | 1.00 | 203 | 1.01 | 4 | 201 |
| e64 | 4 | 1.33 | 246 | 0.89 | 4 | 1.33 | 272 | 0.99 | 3 | 275 |
| 9sym | 5 | 1.00 | 143 | 1.17 | 5 | 1.00 | 124 | 1.02 | 5 | 122 |
| delay ratio | | 1.10 | | | | 1.09 | | | 1 | |
| area ratio | | | | 1.11 | | | | 1.07 | | 1 |

**Table 2. Comparison of *SLDMAP* with *dmig* and *dogma* using *CutMap*** (D: Delay; A: Area; R: Ratio to *SLDMap*)

| Circuit | dmig | | | | dogma | | | | SLDMap | |
|---|---|---|---|---|---|---|---|---|---|---|
| | D | R | A | R | D | R | A | R | D | A |
| Count | 5 | 1.67 | 31 | 0.70 | 5 | 1.67 | 31 | 0.70 | 3 | 44 |
| 9symml | 5 | 1.25 | 120 | 1.24 | 4 | 1.00 | 100 | 1.03 | 4 | 97 |
| frg1 | 5 | 1.25 | 59 | 0.87 | 4 | 1.00 | 74 | 1.09 | 4 | 68 |
| i3 | 4 | 1.33 | 154 | 2.33 | 4 | 1.33 | 138 | 2.09 | 3 | 66 |
| alu2 | 7 | 1.00 | 129 | 1.07 | 7 | 1.00 | 122 | 1.01 | 6 | 122 |
| x1 | 3 | 1.00 | 129 | 1.12 | 3 | 1.00 | 124 | 1.08 | 3 | 115 |
| C432 | 10 | 1.00 | 125 | 1.19 | 10 | 1.00 | 127 | 1.21 | 10 | 105 |
| alu4 | 9 | 1.13 | 217 | 1.06 | 8 | 1.00 | 207 | 1.01 | 8 | 205 |
| rot | 7 | 1.17 | 268 | 0.95 | 7 | 1.17 | 271 | 0.96 | 6 | 275 |
| i2 | 5 | 1.25 | 118 | 1.49 | 4 | 1.00 | 79 | 1.00 | 4 | 79 |
| C880 | 8 | 1.00 | 98 | 0.98 | 8 | 1.00 | 100 | 1.00 | 8 | 100 |
| C2670 | 9 | 1.13 | 235 | 0.86 | 9 | 1.13 | 242 | 0.89 | 8 | 272 |
| dalu | 5 | 1.00 | 357 | 1.17 | 5 | 1.00 | 318 | 1.04 | 5 | 305 |
| C3540 | 10 | 1.00 | 543 | 1.02 | 10 | 1.00 | 541 | 1.01 | 10 | 534 |
| toolarge | 5 | 1.00 | 162 | 0.99 | 5 | 1.00 | 176 | 1.07 | 5 | 164 |
| t481 | 6 | 1.20 | 175 | 1.07 | 6 | 1.20 | 163 | 1.00 | 5 | 163 |
| k2 | 6 | 1.00 | 428 | 0.98 | 6 | 1.00 | 442 | 1.02 | 6 | 435 |
| C7552 | 8 | 1.14 | 541 | 1.07 | 7 | 1.00 | 527 | 1.04 | 7 | 506 |
| des | 5 | 1.00 | 1020 | 1.11 | 5 | 1.00 | 918 | 1.00 | 5 | 919 |
| C499 | 4 | 1.00 | 66 | 1.00 | 4 | 1.00 | 66 | 1.00 | 4 | 66 |
| apex6 | 5 | 1.00 | 248 | 1.02 | 5 | 1.00 | 251 | 1.04 | 5 | 242 |
| apex7 | 4 | 1.00 | 71 | 1.06 | 4 | 1.00 | 72 | 1.07 | 4 | 67 |
| duke2 | 4 | 1.00 | 192 | 1.04 | 4 | 1.00 | 192 | 1.04 | 4 | 185 |
| e64 | 4 | 1.33 | 237 | 0.94 | 4 | 1.33 | 255 | 1.01 | 3 | 253 |
| 9sym | 5 | 1.00 | 132 | 1.13 | 5 | 1.00 | 118 | 1.01 | 5 | 117 |
| delay ratio | | 1.12 | | | | 1.08 | | | 1 | |
| area ratio | | | | 1.10 | | | | 1.06 | | 1 |

**Table 3. Impact of W on depth and area**

| circuit | W=2 | | W=4 | | W=6 | | W=8 | |
|---|---|---|---|---|---|---|---|---|
| | depth | area | depth | area | depth | area | depth | area |
| count | 5 | 31 | 5 | 31 | 3 | 44 | 3 | 49 |
| 9symml | 4 | 97 | 4 | 97 | 4 | 97 | 4 | 98 |
| frg1 | 4 | 68 | 4 | 68 | 4 | 68 | 4 | 70 |
| i3 | 3 | 66 | 3 | 66 | 3 | 66 | 3 | 66 |
| alu2 | 7 | 114 | 7 | 111 | 6 | 122 | 6 | 127 |
| x1 | 3 | 117 | 3 | 117 | 3 | 115 | 3 | 116 |
| C432 | 10 | 105 | 10 | 113 | 10 | 105 | 10 | 105 |
| alu4 | 8 | 205 | 8 | 215 | 8 | 205 | 8 | 245 |
| rot | 7 | 255 | 7 | 258 | 6 | 282 | 6 | 305 |
| i2 | 4 | 79 | 4 | 79 | 4 | 79 | 4 | 79 |
| C880 | 8 | 100 | 8 | 101 | 8 | 100 | 8 | 128 |
| C2670 | 9 | 230 | 9 | 248 | 8 | 272 | 8 | 285 |
| dalu | 5 | 291 | 5 | 298 | 5 | 305 | 5 | 319 |
| C3540 | 10 | 527 | 10 | 534 | 10 | 534 | 10 | 534 |
| too_large | 5 | 159 | 5 | 159 | 5 | 164 | 5 | 163 |
| t481 | 6 | 160 | 6 | 153 | 5 | 163 | 6 | 160 |
| k2 | 6 | 435 | 6 | 450 | 6 | 435 | 6 | 435 |
| C7552 | 7 | 506 | 7 | 523 | 7 | 506 | 7 | 506 |
| des | 5 | 919 | 5 | 926 | 5 | 919 | 5 | 919 |
| C499 | 4 | 66 | 4 | 66 | 4 | 66 | 4 | 66 |
| apex6 | 5 | 239 | 5 | 237 | 5 | 242 | 5 | 243 |
| apex7 | 4 | 72 | 4 | 69 | 4 | 67 | 4 | 73 |
| duke2 | 4 | 182 | 4 | 184 | 4 | 185 | 4 | 207 |
| e64 | 4 | 244 | 4 | 241 | 3 | 253 | 3 | 266 |
| 9sym | 5 | 116 | 5 | 116 | 5 | 117 | 5 | 115 |
| total | 142 | 5383 | 142 | 5460 | 135 | 5511 | 136 | 5680 |
| delay ratio | 1.05 | | 1.05 | | 1 | | 1.01 | |
| area ratio | | 1 | | 1.01 | | 1.02 | | 1.05 |

## 4.1 Mapping for All Possible Decompositions

**Table 1** shows that *SLDMap* consistently produces solutions with better delay than *dmig* and *dogma* when *DAGMap* is used to generate the mapping solution. This confirms Theorem 4, which shows that *SLDMap* algorithm is at least as good as performing *DAGMap* on all decompositions encoded in the mapping graph.

## 4.2 Improvement Over Existing Approaches

**Table 2** shows that *SLDMap* outperforms *dmig* by 12% in delay, 10% in area and *dogma* by 8% in delay, 6% in area when the state-of-the-art FPGA mapping algorithm *CutMap* is performed. The result verifies that the combined approach is better in terms of both delay and area than the separate approach. However, the improvement is not that dramatic, which indicates that the conventional flow is quite close to the optimal result that could possibly be obtained. This mainly comes from two factors: (1) The level of the 2-bounded network is closely related to the level of the mapped network; (2) *CutMap* (an enhancement of *FlowMap*) is capable of generating both optimal depth and small area simultaneously.

## 4.3 Impact of W on Mapping Results

When W increases, delay of the final mapping solutions gets better, but area also increases slightly as shown in **Table 3**. One thing to point out is that, given two different numbers $W_1$ and $W_2$ (assume $W_1 < W_2$), the mapping graphs $G_1$ constructed from the $W_1$-bounded network is not a strict sub-set of the mapping graph $G_2$ constructed from the $W_2$-bounded network. It is possible that some logic decompositions in $G_1$ are not contained in $G_2$, which accounts for the fact that the total depths for $W = 8$ are slightly larger compared with $W = 6$.

## 4.4 Statistics and Runtime

**Table 4** shows some statistics of the mapping graph and *SLDMAP* algorithm. The five columns show the value of W, the number of nodes in the W-bounded network, the number of ugates and cycles in the mapping graph, and the total runtime of *SLDMap* algorithm (in seconds on a Sun Ultra-60 360MHz workstation) respectively.

**Table 4. Statistics about the mapping graph for circuit *rot***

| W | # nodes | # ugates | # cycles | runtime |
|---|---|---|---|---|
| 2 | 1141 | 645 | 4 | 6 |
| 4 | 957 | 1267 | 5 | 10 |
| 6 | 881 | 2875 | 12 | 25 |
| 8 | 843 | 6472 | 94 | 150 |

## 4.5 Scalability

Also, we notice that *SLDMap* is more scalable than the commonly used optimization script *script.rugged*. For most big circuits, *script.rugged* fails, so *script.algebrai*c is used instead. When our algorithm is performed on networks optimized by algebraic scripts, an area reduction of 8% over *dmig* can be achieved as shown in **Table 5**. In general, the new approach is more scalable than *script.rugged* but may not be as scalable as *script.algebraic*.

**Table 5. Comparison with *dmig* and *dogma* after *script.algebraic***

| circuit | *dmig* | | | *dogma* | | | *SLDMap* | |
|---|---|---|---|---|---|---|---|---|
| | D | A | R | A | D | R | D | A |
| C7552 | 7 | 456 | 1.08 | 7 | 462 | 1.09 | 7 | 422 |
| s35932 | 3 | 2659 | 1.11 | 3 | 2630 | 1.10 | 3 | 2387 |
| too_large | 6 | 755 | 1.04 | 6 | 759 | 1.04 | 6 | 727 |
| alu4 | 6 | 1207 | 1.03 | 6 | 1211 | 1.03 | 6 | 1176 |
| total | 22 | 5077 | | 22 | 5062 | | 22 | 4712 |
| ratio | | 1.08 | 1.07 | | 1.07 | 1.07 | | 1 |

## 5. Conclusions and On-going Work

*SLDMap* performs delay-minimized technology mapping on *a large set* of decompositions and simultaneously controls mapping area under the delay constraints. This work shows that the best algorithms in conventional flow (*dmig* + *CutMap*) produce satisfactory depth result even with a fixed decomposition. When structural decomposition and technology mapping are taken into consideration simultaneously, *SLDMap* outperforms the state-of-the-art separate flow (*dmig* + *CutMap)* by 12% in depth and 10% in area on average.

Since BDD is used during the mapping graph construction phase to reduce the mapping graph, *SLDMap* currently cannot handle some BDD-hard circuits, e.g. C6288 (multiplier). The proposed solution for this problem is to build local BDDs for each gate that needs to be decomposed, instead of building a global BDD for the entire network. Although some global functional equivalence and sharing information will be lost, it will enable us to handle BDD-hard circuits and other large circuits.

For big circuits with more than 100K gates, an effective partitioning algorithm shall be developed to divide the problem into smaller sizes.

Another direction to explore is how to better handle networks with W bigger than 10. We may enumerate a large set of *promising* decompositions to limit the mapping graph size and control the construction time. Even for W less than 10, we can still keep only promising decompositions and explore the smaller solution space in a more thorough way or in a shorter amount of time.

We will also investigate the impact of this combined approach after placement and routing on real devices.

## 6. ACKNOWLEDGMENTS

The authors gratefully thank Dr. Yeanyow Huang on usage of *dogm*a and *cutmap* for our experiments. We would also like to thank Dr. Songjie Xu, Chang Wu and Wangning Long for providing tons of help. This work is partially supported by Actel, Lucent Technology and Xilinx under the California Micro Program and the NSF under grant MIP-9357582.

## 7. REFERENCES

[1] J. Cong and Y.-Y. Hwang, "Structural Gate Decomposition for Depth-Optimal Technology Mapping in LUT-based FPGA Design," 33rd ACM/IEEE Design Automation Conference, 1996, pp. 726-729.

[2] R. J. Francis and J. Rose and K. Chung, "Chortle: A Technology Mapping Program For Lookup Table-Based Field Programmable Gate Arrays," 27th ACM/IEEE Design Automation Conference, 1990, pp. 613-619.

[3] R. Murgai, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli, "Performance Directed Synthesis For Table Lookup Programmable Gate Arrays," IEEE International Conference on CAD, 1991, pp. 572-575.

[4] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng and P. Trajmar, "DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization," IEEE Design and Test of Computers, 1992, pp. 7-20.

[5] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems, 1994, vol. 13, no. 1, pp. 1-12.

[6] J. Cong and Y. Ding, "On Area/Delay Trade-off in LUT-based FPGA Technology Mapping," IEEE Trans. on VLSI Systems, June 1994, vol. 2, no. 2, pp. 137-148.

[7] J. Cong and Y.-Y. Hwang, "Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping," ACM 3rd Int'l Symp. on Field Programmable Gate Arrays, 1995, pp. 68-74.

[8] A. Farrahi and M. Sarrafzadeh, "Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping," IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems, 1994, vol. 13, no. 11, pp. 1319-1332.

[9] J. Cong and Y. Ding, "Combinational Logic Synthesis for SRAM Based Field Programmable Gate Arrays," ACM Transactions on Design Automation of Electronic Systems, 1996, vol. 1, no. 2, pp. 145-204.

[10] E. Lehman, Y. Watanabe, J. Grodstein and H. Harkness, "Logic Decomposition during Technology Mapping," IEEE/ACM International Conference on Computer-Aided Design. 1995, pp.264-271.

[11] E. Lehman, Y. Watanabe, J. Grodstein and H. Harkness, "Logic Decomposition during Technology Mapping," IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems, 1997, vol. 16, no. 8, pp. 813-834.

[12] D. Jongeneel, R. Otten, Y. Watanabe and R. K. Brayton, "Area and Search Space Control for Technology Mapping," 37th ACM/IEEE Design Automation Conference, 2000, pp. 86-91.

[13] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41, 1992.

[14] J. Cong, Y. Ding and C. Wu, "Cut Ranking and Pruning: Enabling A General And Efficient FPGA Mapping Solution," ACM 7th Int'l Symp. on Field Programmable Gate Arrays, 1999, pp. 29-35.

[15] G. Chen and J. Cong, "Simultaneous Logic Decomposition with Technology Mapping in FPGA Designs," *UCLA Computer Science Department Technical Report* CSD-200030.

[16] J. Cong, J. Peck and Y. Ding, "RASP: A General Logic Synthesis System for SRAM-based FPGAs," Proc. ACM 4th Int'l Symp. on FPGA, 1996, pp. 137-143. The RASP package can be downloaded from http://cadlab.cs.ucla.edu.

[17] S. Panda, F. Somenzi, and B. F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," IEEE International Conference on Computer-Aided Design, November 1994, pp. 628-631.