FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs

Jason Cong and Yuzheng Ding

Department of Computer Science
University of California, Los Angeles, CA 90024

Abstract

The field programmable gate-array (FPGA) has become an important technology in VLSI ASIC designs. In the past a few years, a number of heuristic algorithms have been proposed for technology mapping in lookup-table (LUT) based FPGA designs, but none of them guarantees optimal solutions for general Boolean networks and little is known about how far their solutions are away from the optimal ones. This paper presents a theoretical breakthrough which shows that the LUT-based FPGA technology mapping problem for depth minimization can be solved optimally in polynomial time. A key step in our algorithm is to compute a minimum height K-feasible cut in a network, which is solved optimally in polynomial time based on network flow computation. Our algorithm also effectively minimizes the number of LUTs by maximizing the volume of each cut and by several post-processing operations. Based on these results, we have implemented an LUT-based FPGA mapping package called FlowMap. We have tested FlowMap on a large set of benchmark examples and compared it with other LUT-based FPGA mapping algorithms for delay optimization, including Chortle-d, MIS-pga-delay, and DAG-Map. FlowMap reduces the LUT network depth by up to 7% and reduces the number of LUTs by up to 50% compared to the three previous methods.

Manuscript received _____. This research is partially supported by a grant from NSF under MIP-9110511, and a grant from Xilinx Inc. under the State of California MICRO program No.92-030. The authors are with the Department of Computer Science, University of California, Los Angeles, CA 90024.

1. Introduction

The short design cycle and low manufacturing cost have made FPGA an important technology for VLSI ASIC designs. The LUT-based FPGA architecture is a popular architecture used by several FPGA manufacturers, including Xilinx and AT&T [15, 28]. In an LUT-based FPGA chip, the basic programmable logic block is a K-input lookup table (K-LUT) which can implement any Boolean function of up to K variables. The technology mapping problem in LUT-based FPGA designs is to cover a general Boolean network (obtained by technology independent synthesis) using K-LUTs to obtain a functionally equivalent K-LUT network. This paper studies the LUT-based FPGA technology mapping problem for delay optimization.

The previous LUT-based FPGA mapping algorithms can be roughly divided into three classes. The algorithms in the first class emphasize on minimizing the number of LUTs in the mapping solutions. This class includes MIS-pga and its enhancement, MIS-pga-new, by Murgai *et al.* based on several synthesis techniques [20, 22], Chortle and Chortle-crf by Francis *et al.* based on tree decomposition and bin packing techniques [11, 14], Xmap by Karplus based on the if-then-else DAG representation [17], the algorithm by Woo based on the notion of edge visibility [27], and the work by Sawkar and Thomas based on the clique partitioning approach [24]. The algorithms in the second class emphasize on minimizing the delay of the mapping solutions. This class includes MIS-pga-delay by Murgai *et al.* which combines the technology mapping with layout synthesis [21], Chortle-d by Francis *et al.* which minimizes the depth increase at each bin packing step [12], and DAG-Map by Cong *et al.* [7, 3] based on Lawler's labeling algorithm. The mapping algorithms in the third class, including that proposed by Bhat and Hill [1], and that by Schlag, Kong, and Chan [26], have the objective of maximizing the routability of the mapping solutions. Although many existing mapping methods showed encouraging results, these methods are heuristic in nature, and it is difficult to determine how far the mapping solutions of these algorithms are away from the optimal solution[1]. It has been of both theoretical and practical interest to CAD researchers to develop optimal FPGA mapping algorithms for general Boolean networks.

This paper presents a theoretical breakthrough which shows that the LUT-based FPGA technology mapping problem for depth minimization can be solved optimally in polynomial time for general K-bounded Boolean networks. A key step in our algorithm is to compute a *minimum height K-feasible cut* in a network, which is solved optimally in polynomial time based on efficient network flow computation. Our algorithm also effectively minimizes the number of

[1] Some previous algorithms achieve optimal mapping for restricted problem domains: Chortle is optimal when the input network is a tree, Chortle-crf and Chortle-d are optimal when the input network is a tree and $K \leq 6$, and DAG-Map is optimal when the mapping constraint is monotone, which is true for trees.

LUTs by maximizing the *volume* of each cut and by several post-processing operations. Based on these results, we have implemented an LUT-based FPGA mapping package named FlowMap. We have tested FlowMap on a set of benchmark examples and compared it with other LUT-based FPGA mapping algorithms for delay optimization, including Chortle-d, MIS-pga-delay, and DAG-Map. FlowMap reduces the LUT network depth by up to 7% and reduces the number of LUTs by up to 50% compared to the three previous methods.

Our result makes a sharp contrast with the fact that the conventional technology mapping problem in library-based designs is NP-hard for general Boolean networks [18, 9]. Due to the inherent difficulty, most conventional technology mapping algorithms decompose the input network into a forest of trees and then map each tree optimally [18, 9]. Such a methodology was also used in some existing FPGA mapping algorithms [11, 14, 12]. However, the result in this paper shows that K-LUT mapping can be carried out directly on general K-bounded Boolean networks to achieve depth-optimal solutions.

The remainder of this paper is organized as follows. Section 2 gives a precise problem formulation and some preliminaries. Section 3 presents our depth-optimal technology mapping algorithm for LUT-based FPGA designs. Section 4 describes several enhancements of our algorithm for area minimization. Experimental results and comparative study are presented in Section 5. Extensions and conclusions are presented in Section 6.

2. Problem Formulation and Preliminaries

A Boolean network can be represented as a directed acyclic graph (DAG) where each node represents a logic gate,[2] and a directed edge (i, j) exists if the output of gate i is an input of gate j. A primary input (PI) node has no incoming edge and a primary output (PO) node has no outgoing edge. We use $input(v)$ to denote the set of nodes which are fanins of gate v. Given a subgraph H of the Boolean network, $input(H)$ denotes the set of *distinct* nodes outside H which supply inputs to the gates in H. For a node v in the network, a *K-feasible cone at v*, denoted C_v, is a subgraph consisting of v and its predecessors[3] such that $|input(C_v)| \leq K$ and any path connecting a node in C_v and v lies entirely in C_v. The *level* of a node v is the length of the longest path from any PI node to v. The level of a PI node is zero. The *depth* of a network is the largest node level in the network. A Boolean network is *K-bounded* if $|input(v)| \leq K$ for each node v.

---

[2] In the rest of the paper *gates* and *nodes* are used interchangeably for Boolean networks.

[3] u is a predecessor of v if there is a directed path from u to v.

We assume that each programmable logic block in an FPGA is a K-input lookup-table (K-LUT) that can implement any K-input Boolean function. Thus, each K-LUT can implement any K-feasible cone of a Boolean network. The technology mapping problem for K-LUT based FPGAs is to cover a given K-bounded Boolean network with K-feasible cones, or equivalently, K-LUTs[4]. Fig. 1 shows an example of mapping a Boolean network into a 3-LUT network. Note that we allow these cones to overlap, which means that the nodes in the overlapped region can be duplicated when generating K-LUTs. In fact, our algorithm is capable of duplicating nodes automatically when necessary, in order to achieve depth optimization. A technology mapping solution S is a DAG in which each node is a K-feasible cone (equivalently, a K-LUT) and the edge (C_u, C_v) exists if u is in $input(C_v)$. Our main objective is to compute a mapping solution that results in the minimum delay.

The delay of an FPGA circuit is determined by two factors: the delay in K-LUTs and the delay in the interconnection paths. Each K-LUT contributes a constant delay (the access time of a K-LUT) independent of the function it implements. Since layout information is not available at this stage, we assume that each edge in the mapping solution contributes a constant delay. In this case, the delay is determined by the depth of the mapping solution, which is known as the *unit delay model*. We say that a mapping solution is *optimal* if its depth is minimum. The primary objective of our algorithm is to find an optimal mapping solution, and the secondary objective is to reduce the number of K-LUTs used in the technology mapping solution.

Several concepts about *cuts* in a network will be used in our algorithm. Given a network $N = (V(N), E(N))$ with a source s and a sink t, a *cut* $(X, \bar{X})$ is a partition of the nodes in $V(N)$ such that $s \in X$ and $t \in \bar{X}$. The *node cut-size* of $(X, \bar{X})$, denoted $n(X, \bar{X})$, is the number of nodes in X that are adjacent to some node in $\bar{X}$, i.e.

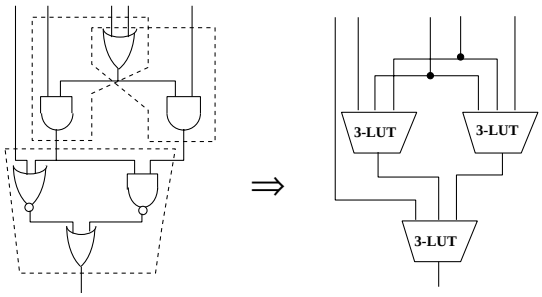

Fig. 1 Mapping a Boolean network to a K-LUT network (K=3).

[4] If the input network is not K-bounded, it may not be covered with K-LUTs directly. In this case, nodes in the network with more than K fanins may have to be decomposed before covering. However, we consider such a decomposition step as part of the synthesis procedure.

$$n(X, \bar{X}) = |\{x : (x, y) \in E(N), x \in X \text{ and } y \in \bar{X}\}|$$

A cut $(X, \bar{X})$ is *K-feasible* if $n(X, \bar{X}) \leq K$. Assume that each edge (u, v) has a non-negative capacity $c(u, v)$. The *edge cut-size* of $(X, \bar{X})$, denoted $e(X, \bar{X})$, is the sum of the capacities of the forward edges that cross the cut, i.e.

$$e(X, \bar{X}) = \sum_{u \in X,\ v \in \bar{X}} c(u, v)$$

Throughout this paper, we assume that the capacity of each edge is one unless specified otherwise. The *volume* of a cut $(X, \bar{X})$, denoted $vol(X, \bar{X})$, is the number of nodes in $\bar{X}$, i.e., $vol(X, \bar{X}) = |\bar{X}|$. Moreover, assume that there is a given label $l(v)$ associated with each node v. The *height* of a cut $(X, \bar{X})$, denoted $h(X, \bar{X})$, is defined to be the maximum label in X, i.e.

$$h(X, \bar{X}) = \max\{l(x) : x \in X\}$$

Fig. 2 shows a cut $(X, \bar{X})$ in a network with given node labels, where $n(X, \bar{X}) = 3$, $e(X, \bar{X}) = 10$, $h(X, \bar{X}) = 2$, and $vol(X, \bar{X}) = 9$.

3. An Optimal LUT-Based FPGA Mapping Algorithm for Depth Minimization

Our algorithm is applicable to any *K-bounded* Boolean network. Given a general Boolean network as input, if it is not K-bounded, there are a number of ways to transform it into a K-bounded network. For example, the Roth-Karp decomposition [23] was used in [20] to obtain a K-bounded network. We use the algorithm DMIG presented in [3], which is based on the Huffman coding tree construction[16], to decompose each multiple input simple gate[5] into a tree of two-input simple gates. According to the result in [3], such a decomposition procedure

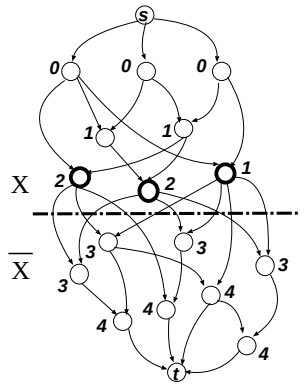

Fig. 2 A 3-feasible cut of edge cut-size 10, volume 9, and height 2.

increases the network depth by at most a small constant factor. The reason for carrying out such a transformation is that if we think of FPGA technology mapping as a process of packing gates into K-LUTs, then, smaller gates will be more easily packed, and the mapping algorithm will be able to pack more gates along critical paths to one K-LUT, resulting smaller depth in the mapping solution. This argument is justified by our experimental results in Table 3 shown in Section 5.

In the rest of this paper, we shall assume that the input networks are K-bounded networks. Although we transform an input network into a network of two-input simple gates, the optimality of our algorithm does not depend on the fact that each node in the given Boolean network is a two-input simple gate. The optimality of our mapping result holds as long as the input network is a K-bounded network, in which the gates need not to be simple.

The fundamental difficulty in the LUT-based FPGA mapping is that the constraint on the number of inputs of a programmable logic block is not a *monotone* clustering constraint. A clustering constraint Γ is *monotone*, if knowing that a network H satisfies Γ implies that any subnetwork of H also satisfies Γ [19]. For example, if we assume that the constraint for each programmable logic block is the number of gates it may cover in the original network, it is a monotone clustering constraint. Unfortunately, limiting the number of distinct inputs of each programmable logic block is not a monotone clustering constraint. For example, Fig. 3 shows a network of three distinct inputs, which is 3-feasible. But the subnetwork consisting of nodes *t*, *v* and *w* has four distinct inputs, which is not 3-feasible. Clustering (or, similarly, mapping) for a monotone clustering constraint Γ is much easier because if a subnetwork H does not satisfy the constraint Γ, we can conclude that H is not a part of *any* cluster. It was shown that Lawler's labeling algorithm [19] can produce a minimum depth clustering solution in polynomial time whenever the clustering constraint is monotone. The DAG-Map algorithm developed by Cong *et al.* [7, 3] modified Lawler's algorithm and applied it to the LUT-based FPGA mapping problem. Although it achieved encouraging results for depth minimization, it was shown that the DAG-

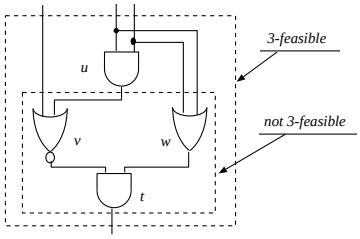

Fig. 3 Constraint on the number of inputs to LUT is not monotone (K=3).

[5] We can always obtain a simple gate network by representing each complex gate in the sum-of-products form and then replacing it with two levels of simple gates.

Map algorithm is not optimal [3].

The mapping algorithm presented in this paper successfully overcomes the difficulty due to the non-monotone clustering constraint in LUT-based FPGA technology mapping. The algorithm runs in two phases. In the first phase, it computes a label for each node which reflects the level of the K-LUT implementing that node in an optimal mapping solution. In the second phase, it generates the K-LUT mapping solution based on the node labels computed in the first phase.

3.1. The Labeling Phase

Given a K-bounded Boolean network N, let N_t denote the subnetwork consisting of node t and all the predecessors of t. We define the *label* of t, denoted $l(t)$, to be the depth of the optimal K-LUT mapping solution of N_t. Clearly, the level of the K-LUT containing t in the optimal mapping solution of N is at least $l(t)$, and the maximum label of all the POs of N is the depth of the optimal mapping solution of N.

The first phase of our algorithm computes the labels of all the nodes in N, according to the topological order starting from the PIs. The topological ordering guarantees that every node is processed after all of its predecessors have been processed. For each PI node u, we assign $l(u) = 0$. Suppose t is the current node being processed. Then, for each node $v \neq t$ in N_t, the label $l(v)$ must have been computed. By including in N_t an auxiliary node s and connecting s to all the PI nodes in N_t, we obtain a network with s as the source and t as the sink. For simplicity we still denote it as N_t. Fig. 4(a) shows part of a Boolean network in which gate t is to be labeled, and Fig. 4(b) shows the construction of the network N_t. Let $LUT(t)$ be the K-LUT that implements node t in an optimal K-LUT mapping solution of N_t. Let $\bar{X}$ denote the set of nodes in $LUT(t)$ and X denote the remaining nodes in N_t. Then, $(X, \bar{X})$ forms a K-feasible cut between s and t in N_t because the number of inputs of $LUT(t)$ is no more than K. Moreover, let u be the node with the maximum label in X, then, the level of $LUT(t)$ is $l(u) + 1$ in the optimal mapping solution of N_t. Recall the definition of the height of a cut in Section 2, we have $h(X, \bar{X}) = l(u)$. Therefore, in order to minimize the level of $LUT(t)$ in the mapping solution of N_t, we want to find a minimum height K-feasible cut $(X, \bar{X})$ in N_t.[6] In other words,

$$l(t) = \min_{(X, \bar{X}) \text{ is } K-\text{feasible}} h(X, \bar{X}) + 1. \quad (1)$$

Based on the above discussion, we have

[6] We exclude the cuts $(X, \bar{X})$ where $\bar{X}$ contains a PI node. Our algorithm to be shown later on guarantees that such kind of cuts are not generated.

Lemma 1 The label $l(t)$ computed by Eq.(1) gives the minimum depth of any mapping solution of N_t. □

Fig. 4(b) and (c) illustrate our labeling method. Since in 4(b) there is a minimum height 3-feasible cut in N_t of height 1, we have $l(t)$=2, and the optimal K-LUT mapping solution of N_t is shown in Fig. 4(c).

There is no existing algorithm for computing a *minimum height* K-feasible cut efficiently. One important contribution of our work is that we have developed an $O(Km)$ time algorithm for computing a minimum height K-feasible cut in N_t, where m is the number of edges in N_t. First, we show that the node labels defined by our labeling scheme satisfy the following property.

Lemma 2 Let $l(t)$ be the label of node t, then $l(t) = p$ or $l(t) = p + 1$, where p is the maximum label of the nodes in $input(t)$.

Proof Let t' be any node in $input(t)$. Then for any cut $(X, \bar{X})$ in N_t, either $t' \in X$, or $(X, \bar{X})$ also determines a K-feasible cut $(X', \bar{X}')$ in $N_{t'}$ with $h(X', \bar{X}') \le h(X, \bar{X})$, where $X' = X \cap V(N_{t'})$ and $\bar{X}' = \bar{X} \cap V(N_{t'})$. If $(X, \bar{X})$ is a minimum height K-feasible cut in N_t, then, in the former case, we have $l(t) = h(X, \bar{X}) + 1 \ge l(t') + 1$, i.e. $l(t) > l(t')$; and in the latter case, we have $l(t') - 1 \le h(X', \bar{X}') \le h(X, \bar{X}) = l(t) - 1$, which implies $l(t) \ge l(t')$. (Note this proves that *the label of any node cannot be smaller than those of its predecessors.*) Therefore, $l(t) \ge p$.

On the other hand, since the network N is K-bounded, $|input(t)| \le K$. Therefore, $(V(N_t) - \{t\}, \{t\})$ is a K-feasible cut. Because each node in $V(N_t) - \{t\}$ is either in $input(t)$ or

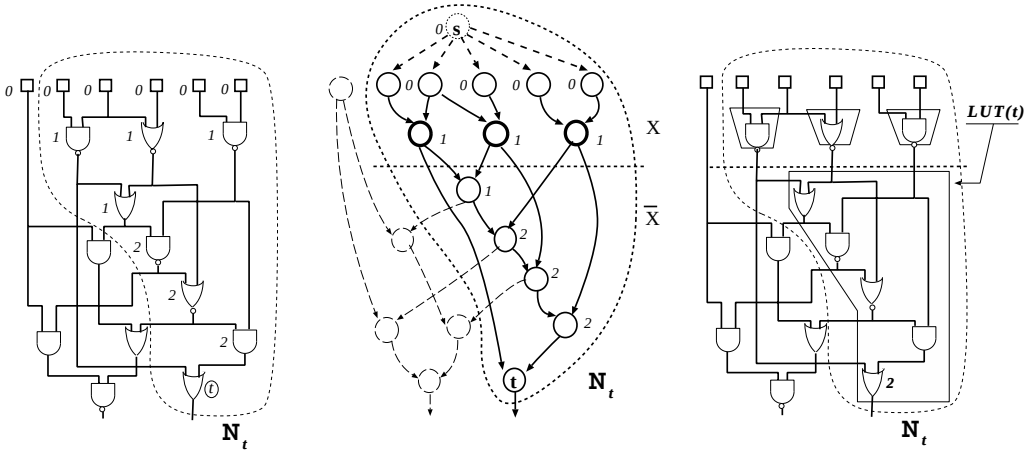

Fig. 4 Computing the label $l(t)$ of node t ($K = 3$).
(a) The partial network; (b) Construction of N_t and the highest 3-feasible cut; (c) Determining $l(t)$.

is a predecessor of some node in $input(t)$, the maximum label of the nodes in $V(N_t) - \{t\}$ is p. Therefore, $h(\, V(N_t) - \{t\},\ \{t\}\,) = p$, i.e. $l(t) \le p + 1$. □

According to Lemma 2, our algorithm first checks if there is a K-feasible cut $(X_t, \bar{X}_t)$ of height $p - 1$ in N_t. If there is such a cut, we assign $l(t) = p$ and node t can be packed with the nodes in $\bar{X}_t$ into one K-LUT in the second phase of our algorithm. Otherwise, the minimum height of the K-feasible cuts in N_t is p and $(V(N_t) - \{t\},\ \{t\})$ is such a cut. In this case, we assign $l(t) = p + 1$ and we shall use a new K-LUT for node t in the second phase.

Whether N_t has a K-feasible cut of height $p - 1$ or not can be tested efficiently using the following method. Let p be the maximum label of the nodes in N_t. We first apply a network transformation on N_t that collapses all the nodes in N_t with label $\ge p$, together with t, into the new sink t'. Denote the resulting network as N'_t, we have the following result.

Lemma 3 N_t has a K-feasible cut of height $p - 1$ if and only if N'_t has a K-feasible cut.

Proof Let H_t denote the set of nodes in N_t that are collapsed into t'.

If N'_t has a K-feasible cut $(\, X', \bar{X}'\,)$, let $X = X'$, and $\bar{X} = (\, \bar{X}' - \{t'\}) \cup H_t$, then clearly $(X, \bar{X})$ is a K-feasible cut of N_t. Since no node in X' $(= X)$ has label p or larger, we have $h(\, X, \bar{X}\,) \le p - 1$. Moreover, according to Lemma 2, $l(t) \ge p$, which implies $h(\, X, \bar{X}\,) \ge p - 1$. Therefore, $h(\, X, \bar{X}\,) = p - 1$.

On the other hand, if N_t has a cut $(\, X, \bar{X}\,)$ of height $p - 1$, then X cannot contain any node of label p or higher. Therefore, $H_t \subseteq \bar{X}$. In this case, $(\, X, (\bar{X} - H_t) \cup \{\, t'\,\}\,)$ forms a K-feasible cut of N'_t. □

For example, Fig. 5(a) shows the network N_t for node t in Fig. 4(a), and Fig. 5(b) shows the induced network N'_t. In order to determine if N'_t has a K-feasible cut, we apply another standard network transformation, called the *node-splitting* transformation, which reduces the node cut-size constraint to an edge cut-size constraint by splitting nodes into edges, so that we can use existing edge cut computation algorithms [10, 8]. Specifically, we construct a network N''_t from N'_t as follows. For each node v in N'_t other than s and t', we introduce two nodes v_1 and v_2 and connect them by an edge (v_1, v_2) in N''_t, which is called a *bridging edge*. The source s and sink t' are also included in N''_t. For each edge (s, v) in N'_t, there is an edge (s, v_1) in N''_t; and for each edge (v, t') in N'_t there is an edge (v_2, t') in N''_t. Moreover, for each edge (u, v) in N'_t $(u \ne s$ and $v \ne t')$, we introduce an edge (u_2, v_1) in N''_t. We assign the capacity of each bridging edge to be one, and the capacity of each non-bridging edge to be infinity. Fig. 5(c) shows the resulting N''_t obtained from N'_t in Fig. 5(b). According to the result in [10] (pp.23-26), we have

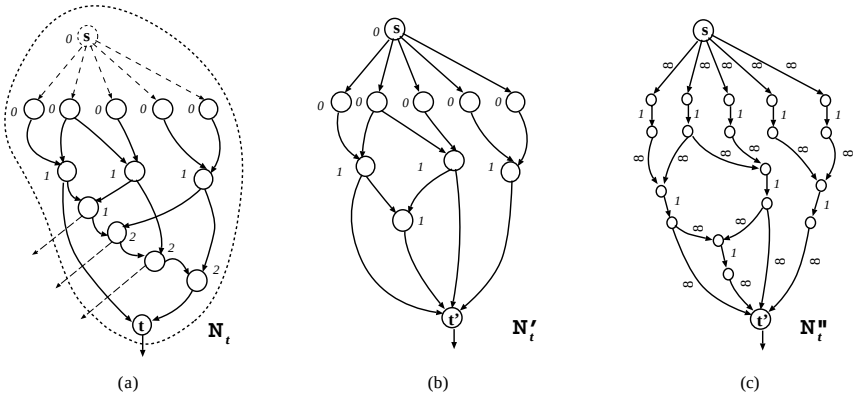

Fig. 5 Network transformations in computing a minimum height K-feasible cut in N_t ($K = 3$).

Lemma 4 N'_t has a K-feasible cut if and only if N''_t has a cut whose edge cut-size is no more than K.

Based on the Max-flow Min-cut Theorem [10, 8], N''_t has a cut whose edge cut-size is no more than K if and only if the maximum flow[7] between s and t' in N''_t has value no more than K. Since we are only interested in testing if the maximum flow is of value K or smaller, we apply the augmenting path algorithm in N''_t to compute a maximum flow. (For the basic concepts of network flow and the details of the augmenting path algorithm, see [10, 8].) Since each bridging edge in N''_t has unit capacity, each augmenting path in the flow residual graph of N''_t from s to t' increases the flow by one unit. If we can find $K + 1$ augmenting paths, then the maximum flow in N''_t has value more than K, and we can conclude that N''_t does not have a cut $(X'', \bar{X}'')$ with $e(X'', \bar{X}'') \le K$. Otherwise, the residual graph is disconnected before we find the $(K + 1)$th augmenting path. We can find a cut $(X'', \bar{X}'')$ of edge-cut size no more than K in N''_t by performing a depth first search starting at the source s, and including in X'' all the nodes which are reachable from s in the residual graph. Since finding an augmenting path takes $O(m)$ time, where m is the number of edges in the residual graph of N''_t (which is in the same order as the number of edges in N_t), we can determine in $O(Km)$ time whether N''_t has a cut of edge cut-size no more than K and find one if such a cut exists. Such a cut $(X'', \bar{X}'')$ in N''_t induces a K-feasible cut $(X', \bar{X}')$ in X'_t, which in turn gives a minimum height K-feasible cut $(X, \bar{X})$ in N_t.[8] Based on the above discussions, we have

---

[7] In this paper, a flow always means a flow from the source to the sink.

[8] It is clear that for the resulting cut $(X, \bar{X})$ in N_t, $\bar{X}$ does not contain any PI nodes since any outgoing edge of the source s in N''_t has infinite capacity.

Theorem 1 A minimum height K-feasible cut in N_t can be found in $O(Km)$ time where m is the number of edges in N_t. □

Applying Theorem 1 to each node in N in the topological order in the labeling phase, we have

Corollary 1 The labels of all the nodes in N can be computed in $O(Kmn)$ time, where n and m are the number of nodes and edges in N, respectively. □

In fact, the result in Theorem 1 can be generalized for computing the minimum height K-feasible cut in a general network with arbitrary node labels and edge capacities as follows.

Theorem 2 Given a general network N with a non-negative integer capacity defined on each edge and a non-negative integer label defined on each node except the sink. For any positive integer K, a minimum height K-feasible cut $(X, \bar{X})$ can be found in $O(m \cdot \min\{K, \sqrt{n}\} \cdot \log L)$ time, and a minimum height cut $(X, \bar{X})$ with $e(X, \bar{X}) \leq K$ can be found in $O(mn\log(n^2/m)\log L)$ time, where n and m are the number of the nodes and edges in N, respectively, and L is the number of different node labels. □

The proof and the detailed algorithm can be found in [4]. This result has been used for delay-optimal K-LUT technology mapping under arbitrary net-delay models [6].

3.2. The Mapping Phase

The second phase of our algorithm is to generate the K-LUTs in the optimal mapping solution. Let L be the set of outputs which are to be implemented using K-LUTs. Initially, L contains all the PO nodes. We process the nodes in L one by one. For each node v in L, assume that $(X_v, \bar{X}_v)$ is the minimum height K-feasible cut in N_v that we have computed in the first phase by the labeling algorithm. We generate a K-LUT v' to implement the function of gate v, using the input signals from X_v to $\bar{X}_v$. That is, the K-LUT v' includes all the gates in $\bar{X}_v$ and $input(v') = input(\bar{X}_v)$. (Since the cut is K-feasible, we have $|input(\bar{X}_v)| \leq K$.) Then, we update the set L to be $(L - \{v\}) \cup input(v')$. It is possible that a gate w belongs to both $\bar{X}_v$ and $\bar{X}_u$ for two different gates v and u in L. In this case, gate w is automatically duplicated and is included in both K-LUTs v' and u'. It is also possible that no K-LUT is generated for a gate w since it has been completely covered by the K-LUTs generated for some of its successors. In general, a K-LUT has to be generated for a gate w if w belongs to $input(v')$ of some K-LUT v' which has been generated, since its output signal is required by v' as input.

The second phase ends when L consists of only PI nodes of the original network. It is clear that at the end of the execution we get a network of K-LUTs which is logically equivalent to the

original network.

Our minimum depth LUT-based FPGA mapping algorithm, named FlowMap, is summarized in Fig 6. Based on the above discussions, we have

Theorem 3 For any K-bounded Boolean network N, the FlowMap algorithm produces a K-LUT mapping solution with the minimum depth in $O(Kmn)$ time, where n and m are the number of nodes and edges in N.

Proof By induction one can easily show that for any node v in N, if a K-LUT v' is generated for v in the second phase, then the level of v' in the mapping solution is no more than $l(v)$, which is the depth of the optimal mapping solution for N_v. Since any mapping solution for N induces a solution for N_v, $l(v)$ is also the best possible depth for the K-LUT generated for v in any mapping solution for N. Therefore, The mapping solution for N generated by the FlowMap algorithm is also optimal. Moreover, since the labeling phase takes $O(Kmn)$ time and the

```
algorithm FlowMap
  /* phase 1: labeling network */
  for each PI node v do
    l(v) := 0;
  T := list of non-PI nodes in topological order;
  while T is not empty do
    remove the first node t from T;
    construct the network N_t;
    let p = max{ l(u) : u ∈ input(t) };
    transform N_t into N'_t by collapsing all nodes in N_t with label p into t;
    transform N'_t into N''_t as follows:
      split every node in { x : x ∈ N'_t, x ≠ s, x ≠ t } into two
        and connect them with a bridging edge of capacity 1;
      assign all non-bridging edges capacity ∞;
    compute a cut (X'', X̄'') in N''_v  s.t.  e(X'', X̄'') ≤ K
      using the augmenting path algorithm;
    if (X'', X̄'') is not found in N''_t then
      X̄_t := {t};   l(t) := p + 1
    else
      induce a cut (X, X̄) in N_t from the cut (X'', X̄'') in N''_t;
      X̄_t := X̄;   l(t) := p
    endif
  endwhile;
  /* phase 2: generate K-LUTs */
  L := list of PO nodes;
  while L contains non-PI nodes do
    take a non-PI node v from L;
    generate a K-LUT v' to implement the function of v
      such that input(v') = input( X̄_v );
    L := (L − { v }) ∪ input(v')
  endwhile
end-algorithm;
```

Fig. 6 Pseudocode of the FlowMap algorithm.

mapping phase takes $O(n+m)$ time, the total complexity of FlowMap is $O(Kmn)$. □

In the current LUT-based FPGA architecture, the typical value of K is 4 or 5. Moreover, if the average number of fanins (or fanouts) of the nodes in N is bounded by a constant (which is two in our implementation), we have $m = O(n)$. Therefore, the complexity of the FlowMap algorithm is $O(n^2)$ in practice.

Note that in a network of n nodes, there are $O(n^K)$ K-feasible cuts. An exhaustive enumeration will resulting in a *pseudo* polynomial time algorithm of complexity $O(n^K)$. Our algorithm, on the other hand, is *strongly* polynomial with respect to K, thus it is much more efficient.

4. Enhancement of the FlowMap Algorithm for Area Optimization

The secondary objective of our technology mapping algorithm is area optimization, i.e. to minimize the number of K-LUTs in the mapping solution. In FlowMap, area optimization is considered by maximizing the volume of each cut during the mapping process and by post-processing operations for K-LUT reduction.

4.1. Maximizing the Cut Volume During Mapping

According to the discussion in the preceding section, for each node t in the input network N, the FlowMap algorithm computes a minimum-height K-feasible cut $(X, \bar{X})$ in N_t and the nodes in $\bar{X}$ will be packed into one K-LUT t' if a K-LUT is generated to implement t. In general, the minimum-height K-feasible cut is not unique. Intuitively, the larger $vol(X, \bar{X}) = |\bar{X}|$ is, the more nodes we can pack into the K-LUT t', and the fewer K-LUTs we use in total. Therefore, our algorithm wants to maximize the volume of the cut during the minimum height K-feasible cut computation.

According to Lemmas 3 and 4, finding a minimum height K-feasible cut $(X, \bar{X})$ in N_t is reduced to finding a K-feasible cut $(X', \bar{X}')$ in N'_t,[9] which is further reduced to finding a cut $(X'', \bar{X}'')$ with $e(X'', \bar{X}'') \le K$ in N''_t. According to the transformations in the preceding section, it is easy to see that $vol(X', \bar{X}') = \frac{1}{2}(vol(X'', \bar{X}'') - e(X'', \bar{X}'') + 1)$, and if the number of nodes with the maximum label in N_t is P, then $vol(X, \bar{X}) = vol(X', \bar{X}') + P$. Note that for a given N_t, P is a constant. Therefore, $vol(X, \bar{X})$ is maximized when $vol(X', \bar{X}')$ is maximized, and $vol(X', \bar{X}')$ is maximized when $vol(X'', \bar{X}'') - e(X'', \bar{X}'')$ is maximized. Thus, we want to find a cut $(X'', \bar{X}'')$ in N''_t such that $e(X'', \bar{X}'') \le K$ and $vol(X'', \bar{X}'') - e(X'', \bar{X}'')$ is maximum. Therefore, we want to

[9] Assume that the minimum height of the K-feasible cuts in N_t is $p - 1$. Otherwise, is $(V(N_t) - \{t\}, \{t\})$ is a minimum height K-feasible cut in N_t, which is trivial to compute.

find a *min-cut* in N''_t (i.e. a cut $(X'', \bar{X}'')$ with the minimum $e(X'', \bar{X}'')$) of the maximum volume.[10] First, we give the following characterization of a maximum volume min-cut.

Lemma 5 There is a unique maximum volume min-cut in any network. Moreover, if $(X, \bar{X})$ is the maximum volume min-cut and $(Y, \bar{Y})$ is another min-cut different from $(X, \bar{X})$, then $X \subset Y$.

Proof Let $(X, \bar{X})$ and $(Y, \bar{Y})$ be two min-cuts of a given network N. Then, $(X \cap Y, \bar{X} \cup \bar{Y})$ is also a min-cut. If $(X, \bar{X})$ is a maximum volume min-cut, we have $|\bar{X}| \geq |\bar{X} \cup \bar{Y}|$, i.e. $\bar{Y} \subseteq \bar{X}$, which implies $X \subseteq Y$. Moreover, if $(Y, \bar{Y})$ is also a maximum volume min-cut, then $\bar{X} = \bar{Y}$, which implies $X = Y$. □

Our algorithm computes a maximum volume min-cut $(X'', \bar{X}'')$ in N''_t as follows: we first compute a maximum flow f in N''_t, and obtain the flow residual graph R_f. We then perform a depth-first search in R_f from the source s to find all the nodes reachable from s. These nodes (including s) form the set X'', while all the other nodes form the set $\bar{X}''$. The next lemma shows that the min-cut computed by our algorithm indeed has the maximum volume.

Lemma 6 Let R_f be the residual graph of a maximum flow f. Let X be the set of nodes in R_f reachable from the source s, and $\bar{X}$ be the set of the remaining nodes. Then, $(X, \bar{X})$ is a maximum volume min-cut.

Proof According to the Max-flow Min-cut Theorem [10, 8], $(X, \bar{X})$ is a min-cut. Let $(Y, \bar{Y})$ be the maximum volume min-cut. According to Lemma 5, If $Y \neq X$, then $Y \subset X$, and there is a node $v \in X - Y$. Since $(Y, \bar{Y})$ is also saturated by f, and $v \in \bar{Y}$, v is not reachable from s, which is a contradiction. □

Combining Theorem 1 and Lemma 6, we have

Theorem 4 A maximum volume min-cut in N''_t can be found in $O(Km)$ time, where m is the number of edges in N_t. □

Therefore, FlowMap maximizes the number of gates covered in each K-LUT by maximizing the volume of each min-cut in N''_t. As a result, area minimization is also achieved during depth-optimal mapping in FlowMap.

---

[10] There is another reason to compute a min-cut (instead of any cut with $e(X'', \bar{X}'') \leq K$) in N''_t: For every signal across the cut in N_t, we need to generate the signal using a K-LUT. Therefore, minimizing the node cut-size in N_t will also lead to reduction of the number of K-LUTs.

4.2. Postprocessing Operations for K-LUT Reduction

In the preceding subsection, we have described the techniques of minimizing the number of K-LUTs during the depth-optimal mapping. After obtaining a K-LUT mapping solution using the FlowMap algorithm, we want to further reduce the number of K-LUTs used in the mapping solution without increasing the depth.

In [3], two depth-preserving operations were developed to minimize the number of K-LUTs in the mapping solutions of DAG-Map. One is called the *predecessor packing*. If a K-LUT u has a fanin K-LUT v, v is fanout-free and $|input(u) \cup input(v)| \leq K$, then v can be merged into u. The predecessor packing operation is shown in Fig. 7. Another operation is called the *gate decomposition*. If a K-LUT u has two fanin K-LUTs v and w, both are fanout-free, and $|input(v) \cup input(w)| \leq K$, then v and w can be merged into one K-LUT that has a single fanout to u, providing that we can carry out the Roth-Karp decomposition [23] on u with respect to its input v and w. The gate decomposition operation is shown in Fig. 8. Clearly, these two operations only take local information into consideration when reducing the K-LUTs.

In this subsection, we want to generalize the idea of predecessor packing so that instead of packing u with one of its fanins into a K-LUT, we try to pack a *set of its predecessors* (including u), denoted P_u, into a single K-LUT. Fig. 9 illustrates the new operation, which is called the *flow-pack*. Clearly, we need to guarantee the condition that $|input(P_u)| \leq K$ in order to carry out the packing. Let M be the current mapping solution and M_u be the subnetwork of M consisting of

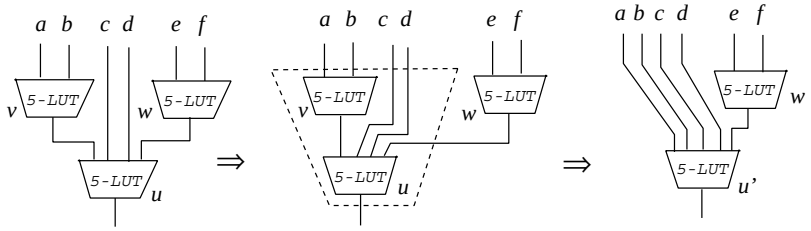

Fig. 7 Predecessor packing (K=5).

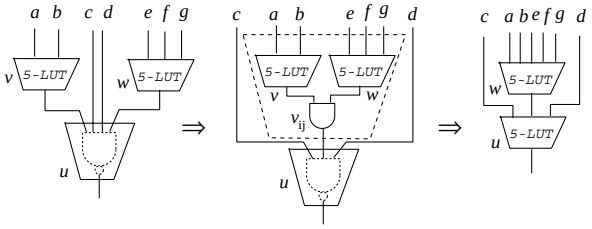

Fig. 8 Gate decomposition (K=5).

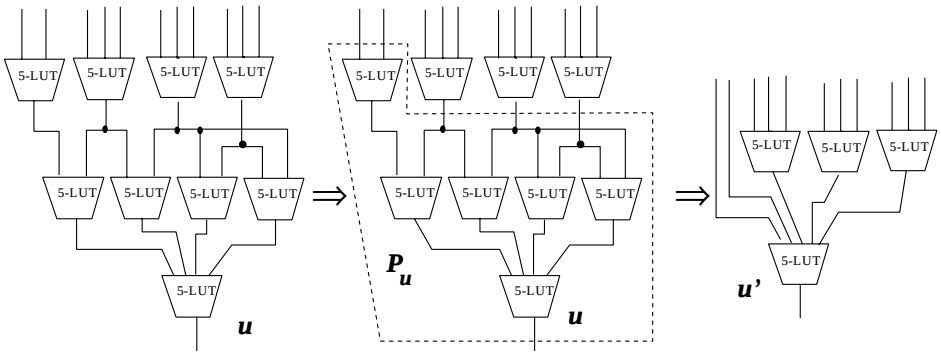

Fig. 9 The flow-pack operation (K=5).

K-LUT u and all its predecessors. Then, P_u can be packed into a single K-LUT if and only if $(V(M_u) - P_u,\ P_u)$ forms a K-feasible cut in M_u. Moreover, the larger $|P_u|$ is, the more K-LUTs we reduce in the mapping solution M. Therefore, we want to find a K-feasible cut with the maximum volume in M_u. The preceding subsection shows that a maximum volume min-cut can be computed efficiently. However, our experimental results showed that using maximum volume min-cut for K-LUT reduction is less effective due to the following reason. The resulting K-LUT network produced by the FlowMap algorithm is usually much denser than the initial 2-input network. Consequently, in many cases, M_u has a unique min-cut ($V(M_u) - \{u\}, \{u\}$). We need to find larger K-feasible cut to overcome this problem.

Since there are only $O(n^K)$ different K-feasible cuts in a network of size n, a maximum-volume K-feasible cut can also be determined in $O(n^K)$ time by enumerating all the K-feasible cuts. However, this method is too expensive when n is large, even for K=5. Therefore, we have developed a heuristic algorithm.

We define the *rank* of a cut $(X, \bar{X})$, denoted $r(X, \bar{X})$, to be an ordered pair $< n(X, \bar{X}), -vol(X, \bar{X}) >$. The cuts can be ordered according to their ranks under the lexicographic ordering, i.e. for any two cuts $(X, \bar{X})$ and $(Y, \bar{Y})$, $r(X, \bar{X}) > r(Y, \bar{Y})$ if $n(X, \bar{X}) > n(Y, \bar{Y})$, or $n(X, \bar{X}) = n(Y, \bar{Y})$ and $vol(X, \bar{X}) < vol(Y, \bar{Y})$. Clearly, the maximum volume min-cut has the smallest rank.

Given a K-LUT network M and a K-LUT u, our algorithm iteratively computes a sequence of cuts in M_u whose rank is monotonically increasing, and at each step, we minimize the increase of the rank of the cut. There are two reasons to minimize the rank. First, we want to limit the increase of the node cut-size at each step since we are interested only in K-feasible cuts. Second, for the cuts of the same node cut-size, the smaller the rank is, the larger volume the cut has.

Specifically, we start with the maximum volume min-cut $(X_0, \bar{X}_0)$, which has the minimum rank. In the ith iteration ($i \geq 1$), we compute a new cut $(X_i, \bar{X}_i)$ from the previous cut $(X_{i-1}, \bar{X}_{i-1})$ in the sequence as follows.

Let $n(X_{i-1}, \bar{X}_{i-1}) = k_{i-1}$, and let $v_1, v_2, \ldots, v_{k_{i-1}}$ be the nodes in X_{i-1} that are adjacent to some node in $\bar{X}_{i-1}$. To compute $(X_i, \bar{X}_i)$, we first collapse all the nodes in $\bar{X}_{i-1}$ into the sink u. Denote the reduced network as M_u^i. Moreover, let $M_u^i(j)$ denotes the network obtained from M_u^i by collapsing v_j $(1 \leq j \leq k_{i-1})$ into the sink u. We compute the maximum volume min-cut $(Y_j, \bar{Y}_j)$ in $M_u^i(j)$ for every j, $1 \leq j \leq k_{i-1}$. Let k_i be the minimum node cut-size of these cuts. Among those cuts that have node cut-size k_i, let $(Y, \bar{Y})$ be the one of maximum volume. Let $X_i = Y$, and $\bar{X}_i = V(M_u) - X_i$, we accept $(X_i, \bar{X}_i)$ as the resulting cut of the ith iteration.

It can be shown [4] that the cut $(X_i, \bar{X}_i)$ has the following properties:

(1) $vol(X_i, \bar{X}_i) > vol(X_{i-1}, \bar{X}_{i-1})$;

(2) $n(X_i, \bar{X}_i) > n(X_{i-1}, \bar{X}_{i-1})$; and

(3) $r(X_i, \bar{X}_i) \leq r(X, \bar{X})$ for any cut $(X, \bar{X})$ such that $X \subset X_{i-1}$.

Therefore, the cut computed at each step is locally optimal.

This iterative procedure ends when at some step l, $(X_l, \bar{X}_l)$ is not K-feasible. The last K-feasible cut, $(X_{l-1}, \bar{X}_{l-1})$, computed in the sequence, is used in the flow-pack algorithm such that $P_u = \bar{X}_{l-1}$. Since the node cut-size is monotonically increasing, we have the number of iterations $l \leq K$. Moreover, each iteration consists of at most K maximum volume min-cut computations, each of time complexity $O(Km)$ (Theorem 4). Therefore, the time complexity for carrying out the flow-pack algorithm at each node is, therefore, bounded by $O(K^3 m)$.

The flow-pack algorithm is implemented as a post-processing step of FlowMap. During the post-processing phase, we first carry out the gate-decomposition operation. (In fact, we carry out a maximum set of gate-decomposition operations simultaneously by computing a maximum cardinality matching among all pairs of gates which are eligible for gate-decomposition. Details of the matching based gate-decomposition algorithm can be found in [3].) Then, we apply the flow-pack operation to each K-LUT u in the mapping solution so that u is collapsed with a maximal subset of its predecessors into a single K-LUT.

The advantage of the flow-pack operation is clear: the flow-pack operation takes the information about the entire subnetwork M_u into consideration, while the predecessor packing examines only the nodes adjacent to u locally. Therefore, in general flow-pack leads to more

substantial reduction of the number of K-LUTs. Our experimental results show that the flow-pack operation alone reduced the number of K-LUTs by 13.5%.

5. Experimental Results

We have implemented the FlowMap algorithm and its preprocessing and post-processing steps using the C language on Sun SPARC workstations. We used input/output routines and general utility functions provided by MIS [2] in our implementation. Given a general Boolean network as input, we first decompose it into a 2-input network of simple gates as described in Section 3. We then apply the FlowMap algorithm to obtain a minimum depth K-LUT mapping solution. Next, we perform a matching-based gate-decomposition procedure on the K-LUT network, followed by the flow-pack operation to reduce the number of K-LUTs in the mapping solution. We chose the size of the K-LUT to be $K = 5$, reflecting, e.g. the XC 3000 FPGA family produced by Xilinx [28]. We tested FlowMap on a number of MCNC benchmark examples and the results were compared with those produced by Chortle-d [13], MIS-pga-delay [21], and DAG-Map [3]. The results are shown in Tables 1 and 2.

Circuit (#gates)		**Chortle-d**		**DAG-Map**		**FlowMap**	
		#LUTs	**depth**	**#LUTs**	**depth**	**#LUTs**	**depth**
5xp1	(104)	26	3	24	3	25	3
9sym	(200)	63	5	61	5	61	5
9symml	(191)	59	5	58	5	58	5
C499	(658)	382	6	207	5	154	5
C880	(548)	329	8	243	8	232	8
alu2	(393)	227	9	169	8	162	8
alu4	(726)	500	10	305	10	268	10
apex6	(779)	308	4	266	4	257	4
apex7	(247)	108	4	91	4	89	4
count	(216)	91	4	81	4	76	3
des	(3263)	2086	6	1433	6	1308	5
duke2	(392)	241	4	192	4	187	4
misex1	(57)	19	2	15	2	15	2
rd84	(141)	61	4	43	4	43	4
rot	(647)	326	6	292	6	268	6
vg2	(120)	55	4	46	4	45	4
z4ml	(48)	25	3	17	3	13	3
total		**4906**	**87**	**3543**	**85**	**3261**	**83**
comparison		**+50.4%**	**+4.8%**	**+8.6%**	**+2.4%**	**1**	**1**

Table 1. Comparison with Chortle-d and DAG-Map.[11]

In Table 1, we used the initial networks provided by Robert Francis which were used by Chortle-d to obtain the results reported in [13] for all the three algorithms. These initial networks were obtained by a sequence of technology independent area and depth optimization steps using MIS. (Since these networks are already 2-input networks, we did not apply our preprocessing algorithm for FlowMap.) Overall, the solutions of Chortle-d used 50.4% more 5-LUTs and had 4.8% larger network depth; the solutions of DAG-Map used 8.6% more 5-LUTs and had 2.4% larger network depth. Note that FlowMap always results in the mapping solution of the smallest depth. Moreover, in terms of the number of 5-LUTs used in the mapping solutions, FlowMap is consistently better than Chortle-d for all examples, and is better than or as good as DAG-Map in most cases.

In Table 2, we cited the results of MIS-pga-delay from [21] since we were unable to run the program directly. The FlowMap results were obtained by first synthesizing the original benchmarks using the MIS optimization script used by Chortle-crf [14] and DAG-Map [3] for technology-independent optimization, then applying the FlowMap algorithm for technology

Circuit	**MIS-pga-delay**		**FlowMap**	
	#LUTs	**depth**	**#LUTs**	**depth**
5xp1	21	2	22	3
9sym	7	3	60	5
9symml	7	3	55	5
C499	199	8	68	4
C880	259	9	124	8
alu2	122	6	155	9
alu4	155	11	253	9
apex6	274	5	238	5
apex7	95	4	79	4
count	81	4	31	5
des	1397	11	1310	5
duke2	164	6	174	4
misex1	17	2	16	2
rd84	13	3	46	4
rot	322	7	234	7
vg2	39	4	29	3
z4ml	10	2	5	2
total	3182	90	2899	84
comparison	+9.8%	+7.1%	1	1

Table 2 Comparison with MIS-pga-delay algorithm.

[11] We also tested the above three algorithms on the input networks used by DAG-Map in [3]. The results showed that compared to FlowMap, DAG-Map used 5.6% more 5-LUTs and had 1.2% larger network depth, while Chortle-d used 52.2% more 5-LUTs and had 10.7% larger network depth. FlowMap produced consistent better results than the other two algorithms.

mapping. Since MIS-pga-delay combines logic synthesis and technology mapping, in several cases it produced mapping solutions of smaller depth than those of FlowMap. However, overall MIS-pga-delay still used 9.8% more 5-LUTs and had 7.1% larger depth.

We have also evaluated the impact of the choices of multi-input gate decomposition methods on the mapping results. We used the DMIG algorithm [3] to decompose the initial networks into two-, three-, four-, or five-input networks and applied FlowMap on the resulting networks. The initial networks for these decomposition algorithms are the same as those used to produce the FlowMap results in Table 2. We summarize the results in Table 3. It can be seen that two-input decomposition gives the best depth results. On the other hand, multi-input decompositions use slightly fewer K-LUTs in some cases. It was observed during the experiments that this reduction is achieved mainly by the post-processing operations. Intuitively, when the gates have larger number of fanins, the mapping phase may result in many unsaturated K-LUTs, which gives more flexibility to the post-processing operations for further reduction of the number of K-LUTs. However, this reduction is usually not worthwhile considering the substantial increase in the depth of the mapping solutions.

Circuit	**DMIG-2**		**DMIG-3**		**DMIG-4**		**DMIG-5**	
	#LUTs	**depth**	**#LUTs**	**depth**	**#LUTs**	**depth**	**#LUTs**	**depth**
5xp1	22	3	22	3	22	3	22	3
9sym	60	5	64	5	67	5	64	6
9symml	55	5	56	5	55	5	56	6
C499	68	4	84	5	84	5	84	5
C880	124	8	108	9	108	9	108	9
alu2	155	9	143	11	146	11	141	13
alu4	253	9	252	11	256	11	263	13
apex6	238	5	238	6	237	6	230	6
apex7	79	4	72	4	69	5	70	5
count	31	5	31	5	31	5	31	5
des	1310	5	1314	6	1343	7	1290	7
duke2	174	4	148	5	144	6	142	6
misex1	16	2	15	2	20	3	20	3
rd84	46	4	46	5	43	5	42	5
rot	234	7	236	8	223	8	222	9
vg2	29	3	29	4	29	3	29	5
z4ml	5	2	5	2	5	2	5	2
total	2899	84	2863	96	2882	99	2819	110
comparison	1	1	-1.2%	+14.3%	-0.6%	+17.9%	-2.7%	+31.0%

Table 3 Impact of decomposition methods on mapping results.

Finally, We have tested the effectiveness of the post-processing phase for area minimization. The results are shown in Table 4. The initial networks used for this experiment are the same as those used in Table 2. The post-processing phase reduced the number of K-LUTs in the mapping solution by 15.1%, and the flow-pack operation alone reduced the number of K-LUTs (after gate-decomposition operation) by 13.5%.

The experiments were carried out on a Sun SPARC IPC workstation (14.8 MIPS). For each benchmark example, our system took less than one minute of CPU time (in most cases a few seconds). Therefore, it is much faster than the Boolean optimization based algorithms.

6. Conclusions and Future Extensions

In this paper, we have presented an technology mapping algorithm named FlowMap for depth minimization in LUT-based FPGA designs, which is optimal for any K-bounded Boolean network. It is based on efficient computation of minimum height K-feasible cuts in a network. A number of area optimization techniques also allow FlowMap to reduce the number of K-LUTs significantly. Compared to the existing LUT-based FPGA technology mapping algorithms for delay optimization, FlowMap reduces the depth of the LUT network by up to 7% and reduces the number of LUTs by up to 50%. FlowMap takes less than one minute of CPU time for each of the benchmarks in our test suite.

One extension is to use a more general delay model other than the unit delay model. For example, Chan, Schlag, and Kong [25] used the *nominal delay model* in FPGA designs where the interconnection delay of a signal net is estimated by the number of fanouts of the net. Their results showed that the nominal delay model estimates the interconnection delay quite well. Based on Theorem 2, we have generalized the FlowMap algorithm to perform delay-optimal mapping under arbitrary net-delay models, including the nominal delay model [6].

Another extension is to combine area and depth optimization in the mapping procedure. Note that the depth of every node is minimum in a FlowMap mapping solution, while in fact only the depths of the nodes on the critical paths need to be minimized to guarantee depth-optimal mapping. The slacks of the non-critical nodes can be utilized for area minimization without affecting the depth optimality. This method can be further extended to the general problem of area minimization under given depth constraint. Based on a set of *depth relaxation* operations defined for non-critical nodes, We have developed an algorithm that can produce a spectrum of area-optimized mapping solutions for different depth constraints, yielding smooth area and depth trade-off in LUT-based FPGA designs [5].

Circuit	Depth	Number of 5-LUTs		
		No Post-processing	Gate-Decomp. Only	Gate-Decomp. & Flow-Pack
5xp1	3	24	24	22
9sym	5	76	71	60
9symml	5	68	64	55
C499	4	80	76	68
C880	8	133	133	124
alu2	9	167	166	155
alu4	9	279	277	253
apex6	5	276	273	238
apex7	4	88	88	79
count	5	43	43	31
des	5	1539	1521	1310
duke2	4	196	193	174
misex1	2	20	18	16
rd84	4	52	50	46
rot	7	258	258	234
vg2	3	32	31	29
z4ml	2	5	5	5
total	84	3336	3291	2899
comparison		1	-1.4%	-15.1%

Table 4 Effectiveness of the FlowMap post-processing phase for area minimization.

The area-optimal mapping problem for LUT-based FPGA designs is still open. Based on the concept of the *maximum fanout-free cones*, in [5] we have developed a polynomial time algorithm for area-optimal K-LUT mapping without node duplication for any fixed *K*.

7. Acknowledgments

We thank Professor Jonathan Rose, Robert Francis, and Rajeev Murgai for their assistance in our comparative study.

References

[1] Bhat, N. and D. Hill, ‘‘Routable Technology Mapping for FPGAs,’’ *First Int’l ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pp. 143-148, Feb. 1992.

[2] Brayton, R. K., R. Rudell, and A. L. Sangiovanni-Vincentelli, ‘‘MIS: A Multiple-Level Logic Optimization,’’ *IEEE Transactions on CAD*, pp. 1062-1081, Nov. 1987.

[3] Chen, K. C., 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, pp. 7-20, Sep. 1992.

[4] Cong, J. and Y. Ding, ‘‘An Optimal Technology Mapping Algorithm fo Delay Optimization in Lookup-Table Based FPGA Designs,’’ in *UCLA Computer Science Department Technical Report CSD-920022*, (May 1992).

[5] Cong, J. and Y. Ding, ‘‘On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping,’’ *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 213-218, June 1993.

[6] Cong, J., Y. Ding, T. Gao, and K. Chen, ‘‘An Optimal Performance-Driven Technology Mapping Algorithm for LUT based FPGAs under Arbitrary Net-Delay Models,’’ *Proc. 1993 Int’l Conf. on CAD and Computer Graphics*, pp. 599-603, Aug. 1993.

[7] Cong, J., A. Kahng, P. Trajmar, and K. C. Chen, ‘‘Graph Based FPGA Technology Mapping For Delay Optimization,’’ *ACM Int’l Workshop on Field Programmable Gate Arrays*, pp. 77-82, Feb. 1992.

[8] Cormen, T., C. Leiserson, and R. Rivest, *Algorithms*, MIT Press, Cambridge, MA (1990).

[9] Detjens, E., G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, ‘‘Technology Mapping in MIS,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 116-119, Nov. 1987.

[10] Ford, L. R. and D. R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, N.J. (1962).

[11] Francis, R. J., J. Rose, and K. Chung, ‘‘Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays,’’ *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 613-619, June 1990.

[12] Francis, R. J., J. Rose, and Z. Vranesic, ‘‘Technology Mapping for Delay Optimization of Lookup Table-Based FPGAs,’’ *MCNC Logic Synthesis Workshop*, 1991.

[13] Francis, R. J., J. Rose, and Z. Vranesic, ‘‘Technology Mapping of Lookup Table-Based FPGAs for Performance,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 568-571, Nov. 1991.

[14] Francis, R. J., J. Rose, and Z. Vranesic, ‘‘Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 613-619, June 1991.

[15] Hill, D., ‘‘A CAD System for the Design of Field Programmable Gate Arrays,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 187-192, June 1991.

[16] Huffman, D. A., ‘‘A method for the construction of minimum redundancy codes,’’ *Proc. IRE 40*, pp. 1098-1101, 1952.

[17] Karplus, K., ‘‘Xmap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 240-243, June 1991.

[18] Keutzer, K., ‘‘DAGON: Technology Binding and Local Optimization by DAG Matching,’’ *Proc. 24th ACM/IEEE Design Automation Conference*, pp. 341-347, 1987.

[19] Lawler, E. L., K. N. Levitt, and J. Turner, ‘‘Module Clustering to Minimize Delay in Digital Networks,’’ *IEEE Transactions on Computers*, Vol. **C-18**(1) pp. 47-57, January 1969.

[20] Murgai, R., Y. Nishizaki, N. Shenay, R. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Logic Synthesis Algorithms for Programmable Gate Arrays,’’ *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 620-625, 1990.

[21] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Performance Directed Synthesis for Table Look Up Programmable Gate Arrays,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 572-575, Nov. 1991.

[22] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Improved Logic Synthesis Algorithms for Table Look Up Architectures ,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 564-567, Nov. 1991.

[23] Roth, J. P. and R. M. Karp, ‘‘Minimization Over Boolean Graphs,’’ *IBM Journal of Research and Development*, pp. 227-238, April 1962.

[24] Sawkar, P. and D. Thomas, ‘‘Technology Mapping for Table-Look-Up Based Field Programmable Gate Arrays,’’ *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pp. 83-88, Feb. 1992.

[25] Schlag, M., P. Chan, and J. Kong, ‘‘Empirical Evaluation of Multilevel Logic Minimization Tools for a Field Programmable Gate Array Technology,’’ *Proc. 1st Int’l Workshop on Field Programmable Logic and Applications*, Sept. 1991.

[26] Schlag, M., J. Kong, and P. K. Chan, ‘‘Routability-Driven Technology Mapping for Lookup Table-Based FPGAs,’’ *Proc. 1992 IEEE International Conference on Computer Design*, pp. 86-90, Oct. 1992.

[27] Woo, N.-S., ‘‘A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 248-251, June 1991.

[28] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, San Jose (1992).