# 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

*In this paper we present a polynomial time technology mapping algorithm, called Flow-Map, that optimally solves the LUT-based FPGA technology mapping problem for depth minimization for general Boolean networks. This theoretical breakthrough makes a sharp contrast with the fact that conventional technology mapping problem in library-based designs is NP-hard. A key step in Flow-Map is to compute a minimum height K-feasible cut in a network, solved by network flow computation. Our algorithm also effectively minimizes the number of LUTs by maximizing the volume of each cut and by several postprocessing operations. We tested the Flow-Map algorithm on a set of benchmarks and achieved reductions on both the network depth and the number of LUTs in mapping solutions as compared with previous algorithms.*

## 1. Introduction

The short design cycle and low manufacturing cost have made FPGA an important technology for VLSI ASIC designs. The LUT-based FPGA is a popular architecture used by several FPGA manufacturers, including Xilinx and AT&T [9, 22]. 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 transform a general Boolean network (obtained by technology independent synthesis) into a functionally equivalent network of K-LUTs. 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 [8, 11, 13, 15, 17, 21]. The algorithms in the second class emphasize on minimizing the delay of the mapping solutions [2, 6, 14]. The algorithms in the third class maximize the routability of the mapping solutions [1, 19]. Although many of the existing mapping methods showed encouraging results, these methods are heuristic in nature, and there is no way to determine how far away the mapping solutions of these algorithms are from the optimal solution in terms of the number of LUTs or the depth of the LUT network.

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 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 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 [4, 12]. Due to this inherent difficulty, most conventional technology mapping algorithms decompose the input network into a forest of trees and then map each tree optimally [4, 12]. Such a methodology was also used in some existing FPGA mapping algorithms [7, 8]. However, our result shows that optimal solutions can be produced efficiently for *general* Boolean network in LUT-based FPGA technology mapping for depth minimization.

## 2. Problem Formulation and Preliminaries

A Boolean network can be represented as a directed acyclic graph (DAG) where each node represents a logic gate, 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 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 such that any path connecting a node in $C_v$ and $v$ lies entirely in $C_v$, and $|input(C_v)| \le K$. 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 $|input(v)| \le K$ for each node $v$.

We assume that each programmable logic block in an FPGA is a 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 Boolean network with K-feasible cones. (Note that we allow these cones to overlap, which means that certain nodes in the original network can be duplicated when generating K-LUTs.) A technology mapping solution $S$ is a DAG where 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 *unit delay model* is used where the delay is determined by the depth of the mapping solution. We say that a mapping solution is *optimal* if its depth is minimum. The main objective of our algorithm is to find an optimal mapping solution; the secondary objective is to reduce the number of K-LUTs used in the solution.

Several important concepts about *cuts* in a network will be used in this paper. 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 $V$ such that $s \in X$ and $t \in \bar{X}$. The *node cut-size* of $(X, \bar{X})$, denoted as $n(X, \bar{X})$, is the number of nodes in $X$ that are adjacent to some node in $\bar{X}$, i.e.

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

A cut $(X, \bar{X})$ is *K-feasible* if its node cut-size is no more than K, i.e., $n(X, \bar{X}) \le K$. Assume that each edge $(u, v)$ has a non-negative capacity $c(u, v)$. Then, the *edge cut-size* of $(X, \bar{X})$, denoted $e(X, \bar{X})$, is the sum of the capacities of the 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 explicitly. 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 $v$. Then, 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. 1 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 [16] was used in [13] to obtain a K-bounded network. In our system, we first transform the given Boolean network into a network of simple gates (i.e. AND, OR, NAND, and NOR gate).[1] Then, we transform the resulting Boolean network into a two-input Boolean network. There are two reasons for carrying out such a transformation. First, we want to limit the number of inputs of each gate to be no more than K so that we do not have to decompose gates during technology mapping. Second, if we think of FPGA technology mapping as a process of packing gates in a given network into K-LUTs, then, intuitively, smaller gates will be more easily packed, with less wasted space in each K-LUT. As proposed in [2, 10, 20], we use an algorithm based on the Huffman coding tree construction to decompose each multiple input simple gate into a tree of two-input simple gates. According to the result in [2], such a decomposition procedure increases the network depth by at most a small constant factor. Although our system transforms the original 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 Boolean network is a two-input simple gate. The optimality of our mapping result holds as long as the input network is K-bounded, in which the gates need not to be simple.

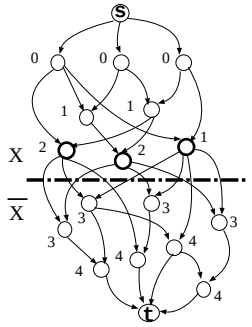



**Fig.1 A 3-Feasible Cut of Height 2**

[1] We represent each complex gate in the sum-of-products form and then replace it with two levels of simple gates.

Our optimal mapping algorithm, named Flow-Map, 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. Due to the length restriction of the paper, the results in this section and next section are stated without proof. The proofs of these results can be found in [3].

### 3.1. The Labeling Phase

Given a K-bounded Boolean network $N$, let $N_v$ denote the subnetwork consisting of node $v$ and all the predecessors of $v$. We define the *label* of $v$, denoted as $l(v)$, to be the depth of the optimal K-LUT mapping solution of $N_v$. Clearly, the level of the K-LUT containing $v$ in the optimal mapping solution of $N$ is at least $l(v)$, 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. For each PI node $v$, we assign $l(v) = 0$. Suppose $t$ is the current node being processed, Then, for each node $v \ne t$ in $N_t$, the label $l(v)$ has 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. 2(a) shows part of a Boolean network in which gate $t$ is being labeled, and Fig. 2(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 $N_t$, and let $\bar{X}$ denote the set of nodes in $LUT(t)$ and $X$ denote the remaining nodes in $N_t$. It is easy to see that $(X, \bar{X})$ 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 $N_t$.[2] In other words,

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

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

There was no known polynomial time algorithm for computing a *minimum height* K-feasible cut. 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 $N_t$.

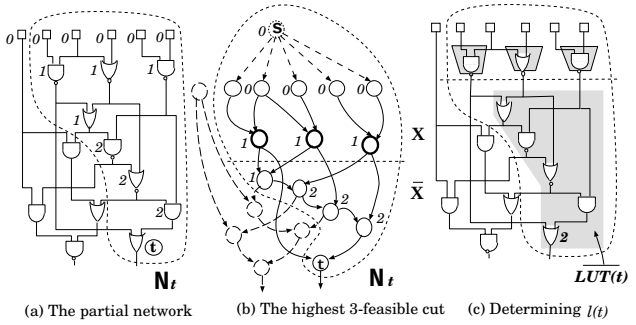
First, we show that the node labels defined by our labeling scheme satisfy the following property.

**Lemma 1** $l(t) = p$ or $l(t) = p + 1$, where $p$ is the maximum label of the nodes in $input(t)$. □

According to Lemma 1, our algorithm first check 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 the same 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 $t$.

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 $\geq p$, together with $t$, into the new sink $t'$. Let $N'_t$ be the resulting network, we have the following result.

**Lemma 2** $N_t$ has a K-feasible cut of height $p - 1$ if and only if $N'_t$ has a K-feasible cut. □



(a) The partial network (b) The highest 3-feasible cut (c) Determining $l(t)$

**Fig.2 Computing the Label of Node (K=3)**

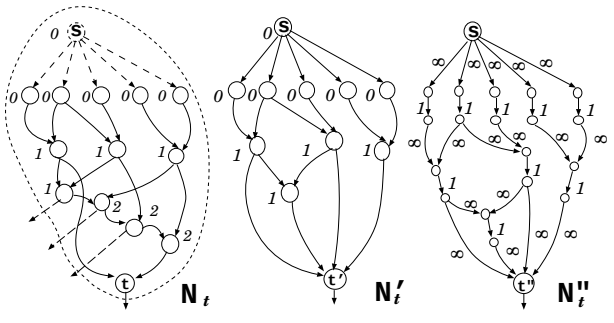
[2] We exclude the cuts $(X, \bar{X})$ where $\bar{X}$ contains a PI node. Our algorithm to be shown later on guarantees that such kind of cuts are not generated.

For example, Fig. 3(a) shows the network $N_t$ for node $t$ in Fig. 2(a), and Fig. 3(b) shows the induced network $N'_t$.

In order to determine if $N'_t$ has a K-feasible cut, we apply another network transformation, which reduces the node cut-size constraint to an edge cut-size constraint by splitting nodes into edges. Specifically, we construct a new 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 $N''_t$ (with $t'$ renamed as $t''$). 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 $N'_t$ ($u \neq s$ and $v \neq t'$), 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. 3(c) shows the resulting $N''_t$ obtained from $N'_t$ in Fig. 3(b).

**Lemma 3** $N'_t$ has a K-feasible cut if and only if $N''_t$ has a cut whose edge cut-size is no more than $K$. □

According to the Max-flow Min-cut Theorem [5], $N''_t$ has a cut whose edge cut-size is no more than $K$ if and only if the maximum flow between $s$ and $t''$ in $N''_t$ has value no more than $K$. We apply the augmenting path algorithm in $N''_t$ to compute a maximum flow. 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 $N''_t$ has a maximum flow of value more than $K$ and we can conclude that $N''_t$ does not have a cut $(X'', \bar{X}'')$ with $e(X'', \bar{X}'') \leq K$. Otherwise, the residual graph is disconnected before we find the $(K + 1)$-th augmenting path, and the disconnected residual graph induces a cut of edge cut-size no more than $K$. Moreover, we can find such a cut $(X'', \bar{X}'')$ by performing a depth first search starting at the source $s$, and including in $X''$ all the nodes which are reachable from $s$. 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.



(a) (b) (c)

**Fig.3 Network Transformations in Minimum Height K-Feasible Cut Computation**

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 $N_t$.[3] Based on the above discussions, we have

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

In fact, the result in Theorem 1 can be generalized to compute the minimum height K-feasible cut in a general network with arbitrary node labels. Details can be found in [3].

### 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 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 $|input(\bar{X}_v)| \le K$.) 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 replicated and is included in both $v'$ 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.

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.

Combining the first and second phases, we have

**Theorem 2** For any K-bounded Boolean network $N$, the Flow-Map 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 $N$. □

In our implementation, $K = 5$ and $m = O(n)$, so the complexity of the Flow-Map algorithm is $O(n^2)$ in practice.

## 4. Enhancement of the Flow-Map Algorithm for Area Optimization

The secondary objective of our technology mapping algorithm is to minimize the number of K-LUTs in the mapping solution. In Flow-Map, this is considered by maximizing the volume of each cut during the mapping process and by postprocessing operations for K-LUT reduction.

### 4.1. Maximizing Cut Volume During Mapping

From the discussion in the preceding section, for each node $t$ in the input network $N$, the Flow-Map algorithm computes a minimum-height K-feasible cut $(X, \bar{X})$ in $N_t$ and the nodes in $\bar{X}$ will be packed into the same K-LUT with $t$ if a K-LUT is generated to implement $t$. In general, 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 a K-LUT, and the fewer K-LUTs we use in total. Therefore, our algorithm wants to maximize the volume of the cut that it finds at each node.

It can be shown that maximizing the volume of a K-feasible cut $(X, \bar{X})$ in $N_t$ is equivalent to maximizing the volume of the corresponding cut $(X'', \bar{X}'')$ in $N''_t$ [3]. Therefore, according to the algorithm presented in the previous section, we want to 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.[4] First, we can show the following results.

**Lemma 4** 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 $X \subset Y$. □

**Lemma 5** 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. □

Combining Theorem 1 and these results, we have

**Theorem 3** 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, Flow-Map maximizes the number of gates covered by each K-LUT by maximizing the volume of each min-cut in $N''_t$. As a result, area minimization is also achieved in the depth optimal mapping of Flow-Map.

### 4.2. Postprocessing Operations for K-LUT Reduction

---

[3] 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.

[4] Since for every signal crossing the cut in $N_t$, we need to generate the signal using a K-LUT, minimizing the node cut-size in $N_t$ will also lead to reduction of the number of K-LUTs. Consequently, we look for a min-cut (instead of any cut with $e(X'', \bar{X}'') \le K$) with maximum volume in $N''_t$.

After obtaining a K-LUT mapping solution using Flow-Map algorithm, we want to further reduce the number of K-LUTs used in the mapping solution without increasing the depth. In [2], two depth-preserving operations were developed to minimize the number of K-LUTs in the mapping solutions of DAG-Map. One is called *predecessor packing* and the other is called *gate decomposition*. Although these operations lead to substantial reduction in the number of K-LUTs, they take only local information into consideration during minimization.

In this subsection, we introduce a new postprocessing operation called *flow-pack*. Given a K-LUT $u$ in the mapping solution, it tries to pack a *set of predecessors* of $u$ (including $u$), denoted $P_u$, into a single K-LUT. (See Fig. 4.) Clearly, we need to guarantee the condition that $|input(P_u)| \le K$ in order to carry out the packing. Let $M$ be the current mapping solution and $M_u$ be the subnetwork of $M$ consisting of K-LUT $u$ and all its predecessors. It is easily seen that $P_u$ 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 $M_u$.

Since no polynomial time optimal algorithm is known for the maximum volume K-feasible cut problem in a network, we have developed a heuristic algorithm to solve this problem. We start with a cut of the minimum node-cut size and maximum volume, which can be computed using the algorithm described in the previous subsection. Then, we gradually increase the volume of the cut (and the node-cut size will increase accordingly as well) until the node-cut size exceeds $K$. The last K-feasible cut in the sequence is recorded as an approximate solution to the maximum volume K-feasible cut problem. The details of this algorithm can be found in [3]. The time complexity of the algorithm was shown to be $O(K^3 m)$.

Based on this approximation algorithm to the maximum volume K-feasible cut problem, the flow-pack operation is implemented as a part of postprocessing step of the Flow-Map package. During the postprocessing phase, we first carry out the matching based gate-decomposition operation as described in [2]. 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.

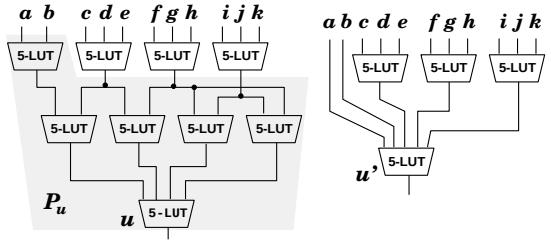



**Fig.4 Flow-Pack Operation (K=5)**

The advantage of the flow-pack operation is clear: the flow-pack operation takes the global information about the entire subnetwork $M_u$ into consideration during the packing process. Therefore, it leads to more substantial reduction of the number of K-LUTs than the predecessor-pack operation defined in [2]. Our experimental results show that on average the postprocessing phase reduced the number of K-LUTs in the mapping solution by 13.0%, and the flow-pack operation alone reduced the number of K-LUTs by 11.6%.

## 5. Experimental Results

We have implemented the Flow-Map algorithm and its preprocessing and postprocessing steps and tested them on a set of MCNC benchmark examples. we chose $K$=5 and compared our results with those produced by previous algorithms including Chortle-d [6], DAG-Map [2], and MIS-pga delay optimization algorithm [14].

Table 1 compares the performance of Flow-Map with Chortle-d and DAG-Map, using the input networks that were used by Chortle-d [6]. 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. Flow-Map always results in the mapping solution of the smallest depth, and in most cases uses less number of 5-LUTs.

We also compared Flow-Map with MIS-pga(delay) in Table 2. The results of MIS-pga(delay) are cited from[14] (since we are unable to run their program directly). We obtain the results of Flow-Map by first synthesizing the original benchmarks using a standard MIS optimization script (used by Chortle-crf [8] and DAG-Map [2]) for technology-independent optimization, then applying the Flow-Map algorithm for technology mapping. Since MIS-pga(delay) combines logic synthesis and technology

| Technology Mapping for 5-LUT FPGAs (I) | | | | | | |
|---|---|---|---|---|---|---|
| | Chortle-d | | DAG-Map | | Flow-Map | |
| | LUTs | depth | LUTs | depth | LUTs | depth |
| *5xp1* | 26 | 3 | 24 | 3 | 25 | 3 |
| *9sym* | 63 | 5 | 61 | 5 | 61 | 5 |
| *9symml* | 59 | 5 | 58 | 5 | 58 | 5 |
| *C499* | 382 | 6 | 207 | 5 | 154 | 5 |
| *C880* | 329 | 8 | 243 | 8 | 232 | 8 |
| *alu2* | 227 | 9 | 169 | 8 | 162 | 8 |
| *alu4* | 500 | 10 | 305 | 10 | 268 | 10 |
| *apex6* | 308 | 4 | 266 | 4 | 257 | 4 |
| *apex7* | 108 | 4 | 91 | 4 | 89 | 4 |
| *count* | 91 | 4 | 81 | 4 | 76 | 3 |
| *des* | 2086 | 6 | 1433 | 6 | 1308 | 5 |
| *duke2* | 241 | 4 | 192 | 4 | 187 | 4 |
| *misex1* | 19 | 2 | 15 | 2 | 15 | 2 |
| *rd84* | 61 | 4 | 43 | 4 | 43 | 4 |
| *rot* | 326 | 6 | 292 | 6 | 268 | 6 |
| *vg2* | 55 | 4 | 46 | 4 | 45 | 4 |
| *z4ml* | 25 | 3 | 17 | 3 | 13 | 3 |
| **total** | 4906 | 87 | 3543 | 85 | 3261 | 83 |
| **cmprsn** | +50.4% | +4.8% | +8.6% | +2.4% | 1 | 1 |

**Table 1. Comparison with Chortle-d and DAG-Map.**[5]

| Technology Mapping for 5-LUT FPGAs (II) | | | | |
|---|---|---|---|---|
| | MIS-pga(delay) | | Flow-Map | |
| | 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 optimization) algorithm.**

mapping, in several cases it produced mapping solutions of smaller depth than those of Flow-Map. However, on average MIS-pga(delay) still used 9.8% more 5-LUTs and had 7.1% larger depth.

Our experiments were carried out on a Sun SPARC IPC workstation (14.8 MIPS). For each benchmark example, the Flow-Map package took only less than a minute of CPU time (a few seconds in most cases) to generate a mapping solution. Therefore, it is much faster than Boolean optimization based algorithms in general.

## 6. Future Work

Currently, we are extending the Flow-Map algorithm to handle more complex delay models (such as the *nominal delay model* [18], which considers both the level and the number of fanouts of each node in delay computation). We are also studying the trade-off between delay and area in technology mapping for FPGA designs.

## 7. Acknowledgments

We thank J. Rose, R. Francis and R. Murgai for their assistance in our comparative study. This research is partially supported by the NSF grant MIP-9110511, the State of California MICRO program NO. 92-030, and a grant from Xilinx Inc..

## 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] 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.

[3] 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).

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] Hoover, H. J., M. M. Klawe, and N. J. Pippenger, ‘‘Bounding Fan-out in Logic Networks,’’ *Journal of Association for Computing Machinery*, Vol. **31**, pp. 13-18, Jan. 1984.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] Wang, A., ‘‘Algorithms for Multi-level Logic Optimization,’’ *U.C.Berkeley Memorandum No. UCB/ERL M89/50*, April 1989.

[21] 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.

[22] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, San Jose, CA (1989).