Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

ON AREA/DEPTH TRADE-OFF IN LUT-BASED FPGA TECHNOLOGY MAPPING

Jason Cong
Yuzheng Ding

October 1992
CSD-920053

On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping

Jason Cong and *Yuzheng Ding*
Department of Computer Science
University of California, Los Angeles, CA 90024

October 23, 1992

Abstract

In this paper, we study the area and depth trade-off in lookup-table (LUT) based FPGA technology mapping. Starting from a depth-optimal mapping solution, we perform a number of depth relaxation operations to obtain a new network with bounded increase in depth so that it is advantageous to subsequent re-mapping for area minimization. We then re-map the resulting network to obtain an area-minimized mapping solution with bounded depth. By gradually increasing the depth bound, for each design we are able to produce a *set* of mapping solutions with smooth area and depth trade-off, while most existing mapping methods produce only a single solution. As the core of the area minimization step, we have developed a polynomial-time optimal algorithm for computing an area-minimum mapping solution without node duplication for a general Boolean network, which makes a significant step towards complete understanding of the general area minimization problem in FPGA technology mapping. The experimental results on MCNC benchmark circuits show that our solution sets outperform the solutions produced by many existing mapping algorithms in terms of both area and depth minimization.

1. Introduction

The field programmable gate array (FPGA) has become a very popular technology in VLSI ASIC design and system prototyping due to its short implementation cycle and low manufacturing cost. An FPGA chip consists of programmable logic blocks, programmable interconnections, and programmable I/O pads. The lookup table (LUT) based FPGA architecture is produced by several FPGA manufacturers [Xi92, Hi91], in which the basic programmable logic block is a K-input lookup table. A K-input LUT (K-LUT) can implement any Boolean function of up to K variables. The technology mapping problem for LUT-based FPGA designs is to convert a general Boolean network into a functionally equivalent K-LUT network.

Previous technology mapping algorithms for LUT-based FPGA design can be roughly divided into three categories according to their optimization objectives. The algorithms in the first category emphasize on minimizing the number of LUTs used in the mapping solution. These algorithms include Chortle-crf [FrRV91a], MIS-pga [Mu90, MuSB91b], XMap [Ka91a], VisMap [Wo91], and TechMap [SaTh92]. The algorithms in the second category emphasize on minimizing the delay of the mapping solution. These algorithms include Chortle-d [FrRV91b], MIS-pga(delay) [MuSB91a], TechMap-L [SaTh92], DAG-Map [CoKT92, ChCD92], and FlowMap [CoDi92]. The algorithms in the third category, including RMap [ScKC92] and the algorithm reported in [BhHi92], emphasize on maximizing the routability of the mapping result. Most of these algorithms are based on heuristic techniques, except FlowMap which guarantees to produce depth-optimal mapping solutions in polynomial time. It remains open if the area-optimal mapping problem for LUT-based FPGAs can be solved in polynomial time.

Although many of the existing algorithms showed encouraging results, they have a common limitation that for a given design, each algorithm produces only a single mapping solution optimized under a fixed objective, while other good mapping solutions under different optimization objectives are ignored. As an example, Figure 1 compares the 5-LUT mapping results by some of the existing algorithms on one of the MCNC benchmark circuits named *rot*. The depth and the number of LUTs of these solutions vary significantly. In general, the area-minimized solutions have much larger depth, while delay-minimized solutions use much more LUTs. However, it is very likely that in practice the best design does not come from one of these two extremes. It is important to let the system designer have the flexibility to choose from a set of

mapping solutions with smooth trade-off between area and depth.

In this paper we study the trade-off between area and depth in LUT-based FPGA technology mapping. Specifically, we are interested in obtaining a *set* of mapping solutions for each design, which can meet various area and depth requirements. In practice, the designer usually has to produce the most compact design satisfying certain depth bound determined by performance specification. To satisfy such a need, our algorithm produces a set of area-minimized mapping solutions under various depth bounds.

The basic approach of our algorithm is as follows. Starting from a depth-optimal mapping solution (computed by the FlowMap algorithm[CoDi92]), we perform a number of depth relaxation operations to obtain a new network with bounded increase in depth so that it is advantageous to subsequent re-mapping for area minimization. We then re-map the resulting network to obtain an area-minimized mapping solution with bounded depth. By gradually increasing the depth bounds, for each design we are able to produce a *set* of mapping solutions with smooth area and depth trade-off. As the core of the area minimization step, we have developed a polynomial-time algorithm for computing an area-optimal mapping solution without node duplication for a general Boolean network, which makes a significant step towards complete understanding of the general area optimization problem in FPGA technology mapping.

We have tested our algorithm on the MCNC benchmark circuits and obtained very encouraging results. For most circuits we are able to produce a set of mapping solutions with smooth area and depth trade-off. At one end, we are able to produce depth-optimal solutions that

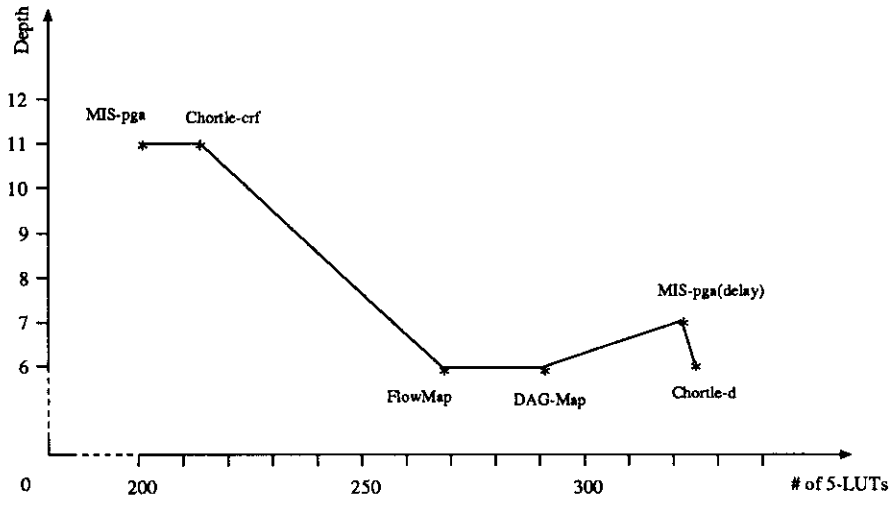


Figure 1 Mapping solutions of various algorithms for *rot* (K=5).

use smaller area than the existing depth minimization mapping algorithms, including Chortle-d, MIS-pga(delay), and FlowMap. At another end, we are able to produce solutions with both smaller area and smaller depth compared to the existing area minimization mapping algorithms, including Chortle-crf and MIS-pga.

The remainder of this paper is organized as follows. Section 2 formulates the problem and introduces several concepts and definitions. Section 3 presents an overview of our algorithm. In Sections 4 and 5, the details of the two phases of our algorithm, i.e. depth relaxation and area minimization, are discussed. Section 6 presents the experimental results. Conclusions and future extensions are presented in Section 7.

2. Problem Formulation

A general Boolean network can be represented as a directed acyclic graph 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 nodes which are the fanins of node v, and $output(v)$ to denote the set of nodes which are the fanouts of node 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. The *level* (or *depth*) 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. In the rest of this paper, we consider only K-bounded networks.[1]

For a node v in the network, a *cone of* v, denoted C_v, is a subgraph of logic gates (excluding PIs) consisting of v and its predecessors[2] such that any path connecting a node in C_v and v lies entirely in C_v. We call v the *root* of C_v. A *fanout-free cone (FFC) at* v, denoted FFC_v, is a cone of v such that for any node $u \neq v$ in FFC_v, $output(u) \subseteq FFC_v$. A *K-feasible cone of* v is a cone C_v such that $|input(C_v)| \leq K$.

We assume that each programmable logic block in an FPGA is a K-input 1-output lookup-table (K-LUT) that can implement any Boolean function of up to K variables. Thus, each K-LUT can implement (or *cover*) any K-feasible FFC in a Boolean network. If a K-LUT LUT_v implements a K-feasible FFC of v, we say that LUT_v *implements* node v and that v is the *root* of LUT_v. If the K-feasible cone C_v is not fanout free, we have to duplicate the non-root nodes in C_v

[1] If a network is not K-bounded, there are a few algorithms to transform it in to a K-bounded network. For example, the DMIG algorithm in [ChCD92] transforms a general network of simple gates into a K-bounded network with minimum depth.

[2] Node u is a predecessor of node v if there is a directed path from u to v.

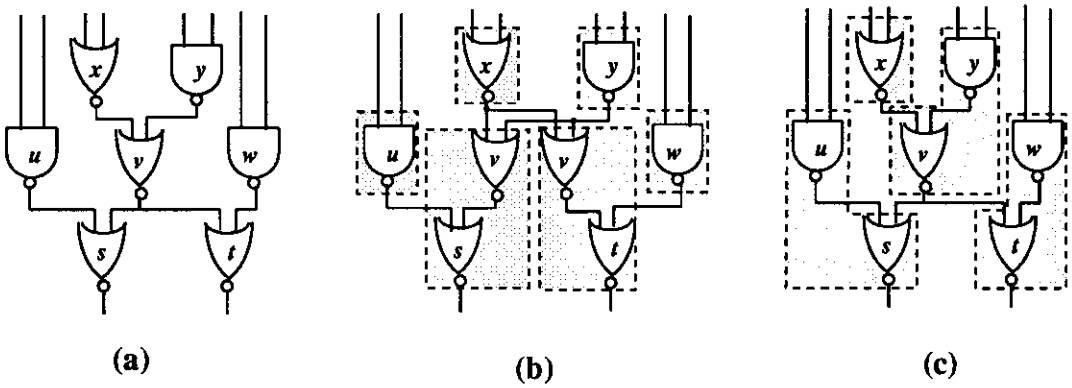


Figure 2 Technology mapping for LUT-base FPGA (K=3).
(a) original network; (b) mapping with node duplication; (c) mapping without duplication.

that have fanouts outside of C_v in order to cover C_v by a K-LUT. Given a K-bounded network, the *technology mapping problem* for K-LUT based FPGA designs is to cover the network with K-feasible FFCs (possibly with node duplications). A technology mapping solution S is a directed acyclic graph where each node is a K-feasible FFC (equivalently, a K-LUT) and the edge (C_u, C_v) exists if u is in $input(C_v)$. Figure 2 shows a Boolean network and two mapping solutions, one with node duplication and the other without node duplication.

We say an LUT mapping solution satisfies the *depth bound D* if the depth of the LUT network is no more than D. Given a depth bound D, the *slack* on node v is defined as follows: If v is not a PI or PO, the slack of v is $D-(L_v+P_v)$, where L_v is the level of v in the network, and P_v is the length of the longest path from v to any PO node. If v is a PI or PO, the slack of v is zero. A node is *critical* if it has zero slack. A path from a PI to a PO consisting of only critical nodes is a *critical path.*

3. Basic Operations and Outline of the Algorithm

In this section, we shall discuss the effect of depth relaxation and node duplication, which are two important factors in determining the area and depth trade-off. Then, we shall give an overview of our algorithm. First, we briefly describe the FlowMap algorithm, which we use to compute a depth-optimal mapping solution as our starting point.

3.1. The FlowMap Algorithm

FlowMap [CoDi92] is an LUT-based FPGA technology mapper that produces depth-optimal mapping solutions for general Boolean networks in polynomial time. The basic idea of

the FlowMap algorithm is to find a depth-optimal mapping for each node in the network, according to the topological order starting from the PI nodes. The depth-optimal mapping of a node v is achieved by computing a *minimum height K-feasible cut* in the subnetwork consisting of all the transitive fanins of v. It was shown that such a cut can be computed in polynomial time. It worths noticing that in a FlowMap mapping solution, every node (LUT) has the minimum possible depth.

3.2. Effect of Depth Relaxation

Insisting minimum depth for every node, including the non-critical ones, may lead to inefficient use of LUTs. Figure 3(a) shows a Boolean network. The mapping solution by FlowMap is shown in Figure 3(b). Another solution is shown in Figure 3(d), which has the same depth as the one in (b) but uses one fewer LUT. Note that LUT_v in (b) has the minimum depth. However, since it is not critical, LUT'_v does not have the minimum depth in (d). In fact, solution (d) can be obtained from (b) by decomposing LUT_v to exclude gate w, as shown in (c), and then pack LUT_u into LUT_v. Since the decomposition increases the depth of the LUT_v, we call it a *depth relaxation* operation. When LUT_v is not critical, this operation does not increase the depth of the network. Depth relaxation is discussed in detail in Section 4.

3.3. Effect of Node Duplication

Node duplication is performed when we use an LUT to cover a K-feasible cone C which has a non-root node with a fanout node outside of C (see Figure 2(b)). In general, node duplication is very important to depth optimization. Without node duplication, we may have to

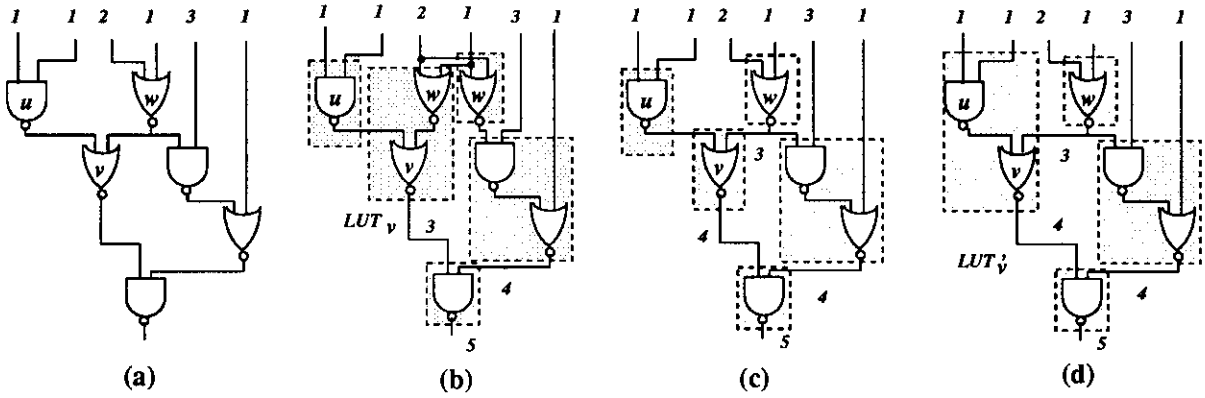


Figure 3 Depth relaxation for area reduction (K=3). The numbers indicate node levels.
(a) original network; (b) solution of FlowMap; (c) after depth relaxation; (d) after re-mapping for area minimization.

implement many multi-fanout nodes explicitly with LUTs, which may lead to large depth in the mapping solution. In the FlowMap mapping solutions, node duplication is heavily used to guarantee the optimal depth. For example, in the mapping solution of the MCNC benchmark circuit *rot*, 90% of the multi-fanout nodes are duplicated. However, node duplication may not be very beneficial to area minimization. If we make m duplications of a node, we need to cover this node by m LUTs, and it may use certain input capacity of each LUT. Therefore, excessive node duplication will very likely result in large number of LUTs. (See Figure 2 for an example.)

In our algorithm, node duplication is automatically carried out by FlowMap for depth minimization. In each of the subsequent depth relaxation steps, we try to eliminate unnecessary node duplications for non-critical nodes, until all the remaining duplications are necessary to satisfy the depth bound. Then, we carry out re-mapping for area minimization without introducing new node duplications. Finally, we apply two post-processing operations that allow necessary node duplications for further area reduction.

3.4. Overview of the Algorithm

Our algorithm starts with the depth-optimal mapping solution produced by FlowMap. For each given depth bound of the mapping solution, our algorithm consists of two phases. During the first phase, we apply a number of depth relaxation operations to produce an intermediate network for subsequent area minimization. In the second phase, we carry out re-mapping for area minimization on the intermediate network. First, we use the DF-Map procedure to compute an area-optimal mapping solution without node duplication. The details of DF-Map will be presented in Section 5. Then, we carry out two post-processing procedures which allow necessary node duplications for further area minimization. The two procedures are MP-Pack, a multi-fanout predecessor packing procedure from the DAG-Map package [ChCD92], and Flow-Pack, a flow-based area minimization procedure from the FlowMap package [CoDi92].

To generate a set of mapping solutions, we gradually increase the depth bound for the mapping solution and repeat the two-phase process for each depth bound. The algorithm stops when no improvement on area is available by further increase of the depth bound. Clearly, the number of iterations is bounded by the depth of the original network. Our algorithm, named FlowMap-r, is outlined as follows.

```
algorithm  FlowMap-r
  call FlowMap to produce a depth-optimal mapping solution;
  repeat
   /* phase 1:  depth relaxation */
   compute slacks;
   while there are nodes with non-zero slacks do
    select a node with non-zero slack;
    apply a depth relaxation operation to decompose the node;
    recompute slacks;
   end-while;
   /* phase 2:  area minimization */
   call FFC-Map to perform area-optimal mapping  without node duplication;
   call MP-Pack to perform matching based  predecessor packing with node duplication;
   call Flow-Pack to perform maximum volume packing with node duplication;
   output mapping solution;
   increase the depth bound by 1;
  until no improvement in area reduction;
end-algorithm.
```

4. Depth Relaxation

Given a non-critical LUT LUT_v rooted at a node v and some node $w \in LUT_v$, the depth relaxation operation decomposes LUT_v into LUT'_v and LUT_w, so that LUT_w becomes a fanin of LUT'_v. In the case where w is a duplicated node in LUT_v and LUT_w already exists in the mapping solution, the depth relaxation simply replaces LUT_v with LUT'_v and let LUT_w be a fanin of LUT'_v, as in Figure 3(c). Otherwise, LUT_w needs to be created explicitly. Normally, we will choose v and w in such a way that after the depth relaxation operation, LUT'_v and LUT_w can be packed with existing LUTs during subsequent re-mapping. Since the depth relaxation operation may lead to different results when applied on different LUTs, we want to apply it to the more promising LUTs first. Figure 4 illustrates three types of depth relaxation, which are considered in our algorithm.

In Figure 4(a), LUT_v contains a duplication of node w, and LUT_w is already in the network. If we apply depth relaxation operation on LUT_v to exclude w, no new LUT needs to be created. Moreover, the input size of LUT_v will be reduced in most cases, so that it may be packed with other LUTs. In this example, LUT_v can be packed either with LUT_u or with LUT_z. Furthermore, elimination of the duplication w also reduces the fanout size of the fanin LUTs of w, which may either enable further packing of LUT_w with the fanin LUT (in this example, LUT_y), or the elimination of a redundant duplication of the fanin node (in this example, node x).

In Figure 4(b), the two duplications of node w are in LUT_u and LUT_v. Since LUT_w needs to be explicitly created, this case is not as favorable as case (a). However, By applying depth relaxation on LUT_v, the input size of LUT_v is reduced, therefore further packing may be possible.

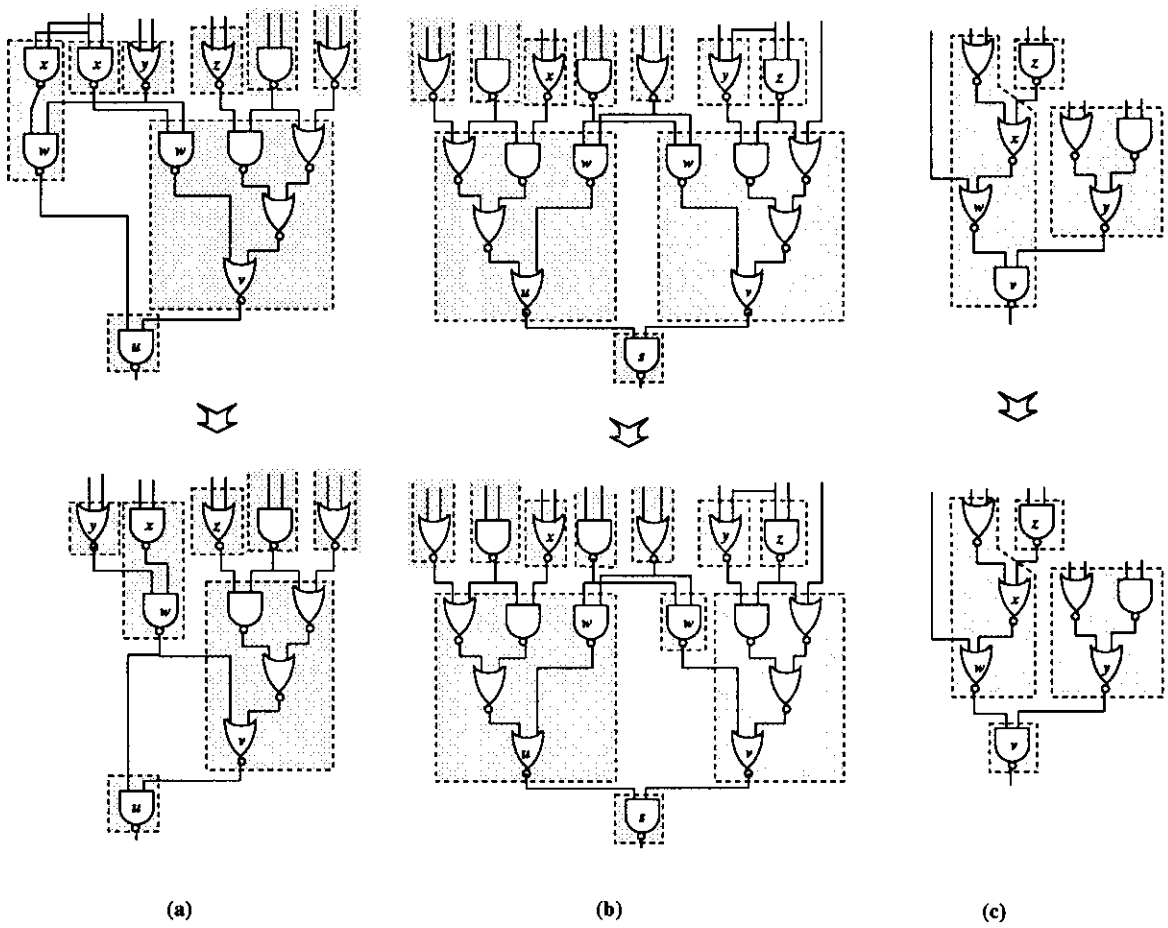


Figure 4 Three types of depth relaxation operations (assume K=5 and LUT_v has non-zero slack).

In this example, LUT_v can be packed with LUT_s, or with LUT_y and LUT_z. Moreover, if we can later apply depth relaxation on LUT_u, no new LUT will be generated for node *w*.

In Figure 4(c), LUT_v contains node *w* which has single fanout. However, decompose LUT_v to exclude *w* may lead to further packing to merge LUT_v with LUT_y, and to merge LUT_w with LUT_z. In this case the depth relaxation is also applicable.

In general, the potential of area reduction after a depth relaxation operation varies. Our algorithm always chooses the operation which will result in the most reduction. After the depth relaxation operation, the slacks of *related* nodes are recomputed, and the process is repeated until no slack is available. Note that the re-mapping is not performed immediately after a single depth relaxation operation. It is invoked after all slacks are exhausted under the current depth bound so that it can perform global optimization for area minimization.

The computational cost of this phase consists of the cost of slack computation for each LUT and the cost of computing potential reduction of each eligible depth relaxation operation. Slack computation takes linear time. Each potential is computed locally and takes constant time. Therefore, the total cost of the depth relaxation procedure for each depth is no more than $O(n^2)$.

5. Area Optimal Mapping without Node Duplication

In this section we present a polynomial time algorithm for an area-optimal mapping without node duplication (*duplication-free mapping*, or *DF-mapping*) in general Boolean networks, which is the core of the re-mapping phase for area minimization. Note that DF-mapping is not equivalent to tree-based mapping. Figure 5 shows a simple example where the optimal DF-mapping uses 2 LUTs, while the optimal tree-based mapping uses 6 LUTs. Our algorithm is based on an important concept called the *maximum fanout free cone*.

5.1. Maximum Fanout Free Cone

The *maximum fanout free cone (MFFC) of* v, denoted $MFFC_v$, is an FFC of v such that for any non-PI node w, if $output(w) \subseteq MFFC_v$, then $w \in MFFC_v$. Figure 6 shows the MFFC of each node (the smallest shadowed area) in a network. Clearly, MFFC is unique for every node, and any FFC of v is contained in $MFFC_v$. Moreover, MFFC has the following important properties.

Lemma 1 If $w \in MFFC_v$, then $MFFC_w \subseteq MFFC_v$.

Proof For any node $u \in MFFC_w$, if there is a path from u to a PO node that does not pass w, let w' be the last node in $MFFC_w$ along the path, then $output(w') \notin MFFC_w$, which contradicts the assumption that $w' \in MFFC_w$. Therefore, every path from u to a PO node must pass w.

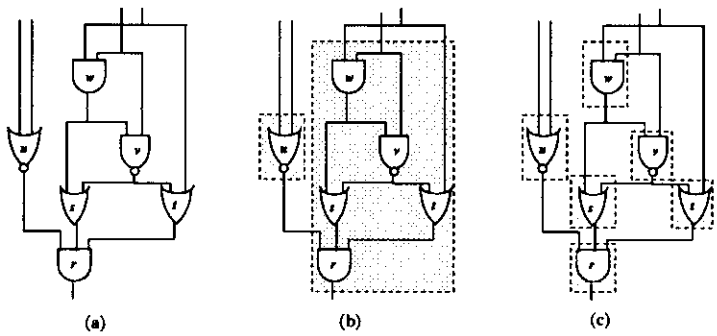


Figure 5 Duplication-free mapping vs. tree-based mapping (K=3).
(a) original network; (b) duplication-free mapping; (c) tree-based mapping.

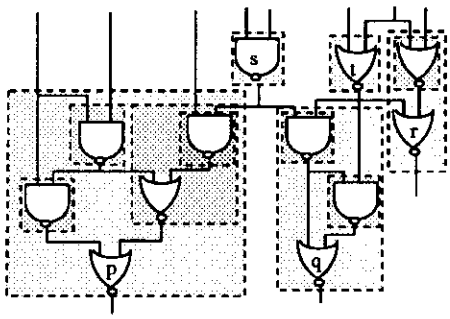


Figure 6 Maximum fanout free cones.

Similarly, since $w \in MFFC_v$, every path from w to a PO node must pass v. This implies that every path from u to a PO node must pass v, so $output(u) \subseteq MFFC_v$, i.e. $u \in MFFC_v$. Therefore, $MFFC_w \subseteq MFFC_v$. □

Lemma 2 Two MFFCs are either disjoint or one must contain another.

Proof If $MFFC_v$ and $MFFC_w$ are not disjoint, let $u \in MFFC_v \cap MFFC_w$. Then, every path from u to a PO node must pass both v and w. Assume that a path from u first passes w then passes v. Then, every path from w to a PO node must also pass v. This implies $w \in MFFC_v$, and according to Lemma 1, $MFFC_w \subseteq MFFC_v$. □

Lemma 3 If LUT_w is in a DF-mapping solution S, then for any v, node $w \in MFFC_v$ implies $LUT_w \subseteq MFFC_v$.

Proof Since there is no node duplication, it is clear that LUT_w implements an FFC rooted at w. Therefore, $LUT_w \subseteq MFFC_w$. Since $w \in MFFC_v$, according to Lemma 1 we know $MFFC_w \subseteq MFFC_v$. Thus, $LUT_w \subseteq MFFC_v$. □

These properties of MFFC allows us to carry out optimal DF-mapping efficiently.

5.2. MFFC Partitioning of General Network

First, we show that a general Boolean network can be decomposed into a set of disjoint MFFCs such that the optimal DF-mapping for the entire network can be carried out in each MFFC independently.

Theorem 1 Let v be a PO node of a general Boolean network N. Then, any optimal DF-mapping solution S of N also induces an optimal DF-mapping solution S_v of $MFFC_v$.

Proof For any LUT_u in S, if $u \in MFFC_v$, according to Lemma 3, $LUT_u \subseteq MFFC_v$. If $u \notin MFFC_v$, since v is a PO, $v \notin MFFC_u$. Then, according to Lemma 2, $MFFC_u \cap MFFC_v = \varnothing$, which implies $LUT_u \cap MFFC_v = \varnothing$. Therefore, any LUT in S is either contained in $MFFC_v$ or disjoint with it. As a result, S induces a mapping solution S_v in $MFFC_v$. Moreover, S_v is also optimal (otherwise S can be further improved). □

According to Theorem 1, we can partition the network N into $MFFC_v$ and $N - MFFC_v$ for any PO node v. An optimal DF-mapping solution consists of an optimal DF-mapping solution of $MFFC_v$ and an optimal DF-mapping solution of $N - MFFC_v$. By applying this theorem recursively on $N - MFFC_v$, we can partition the entire network N into a set of disjoint MFFCs so that we can compute the optimal DF-mapping for each MFFC independently to obtain an optimal DF-mapping solution of N. In Figure 6, the MFFCs of nodes p, q, r, s, and t form a disjoint partition of the network. In the next subsection we shall discuss how to compute an optimal DF-mapping for an MFFC.

5.3. Optimal DF-Mapping for MFFCs

Assume that we want to compute an area-optimal[3] DF-mapping solution of $MFFC_v$. First, we introduce some basic concepts about *cuts* in $MFFC_v$.

A *cut* of $MFFC_v$ is a partition $(X, \bar{X})$ of $MFFC_v$ such that $\bar{X}$ is an FFC of v. The *size* of a cut $(X, \bar{X})$ is defined to be $| input(\bar{X}) |$. A cut is *K-feasible* if its size is no more than K. Clearly, a cut $(X, \bar{X})$ of $MFFC_v$ is K-feasible if and only if $\bar{X}$ can be covered by a K-LUT rooted at v.

For each K-feasible cut $P = (X, \bar{X})$ of $MFFC_v$, we can cover $\bar{X}$ with a K-LUT LUT_v^P, and partition $X = MFFC_v - \bar{X}$ into a set of disjoint MFFCs $MFFC_{v_1^P}$, $MFFC_{v_2^P}$, ..., $MFFC_{v_m^P}$. Then, we recursively compute the area-optimal DF-mapping of each $MFFC_{v_i^P}$ $(1 \le i \le m)$. The *cost* of the cut P is defined to be $cost(P) = 1 + \sum_{i=1}^{m} area(MFFC_{v_i^P})$, where $area(MFFC_{v_i^P})$ is the area of the area-optimal DF-mapping of $MFFC_{v_i^P}$. Clearly, $cost(P)$ gives the area of the best DF-mapping solution of $MFFC_v$ if $\bar{X}$ is covered by LUT_v^P. Therefore, We generate each K-feasible cut of $MFFC_v$ and choose the cut with least cost. Each cut cost computation involves recursively solving a set of DF-mappings for MFFCs of smaller sizes.

It is not difficult to see that there are only polynomial number of K-feasible cuts, since the total number of possible combinations of K or fewer nodes is $O(n^K)$, where n is the number of

[3] It is easy to see that the discussion can be applied to depth-optimal DF-mapping by simply altering the cost function.

nodes in the MFFC. In practice, however, examining all these combinations to compute the K-feasible cut with least cost is too expensive, since most of them do not form a K-feasible cut. In the following two subsections we present a more efficient algorithm to generate the K-feasible cuts in $MFFC_v$.

For simplicity of the discussion, in the remainder of this section, we represent a cut $(X, \bar{X})$ by a string $v_1 v_2 \cdots v_m$, where $input(\bar{X}) = \{v_1, v_2, ..., v_m\}$. For our purpose, the order of the nodes in the string is irrelevant, e.g. $v_1 v_2 \cdots v_m = v_2 v_1 \cdots v_m$. Moreover, we define the operator * on two cuts to be the concatenation of the two corresponding strings, i.e., $v_1 v_2 \cdots v_m * u_1 u_2 \cdots u_n = v_1 v_2 \cdots v_m u_1 u_2 \cdots u_n$.

5.3.1. Cut Generation for Trees

Assume that $MFFC_v$ is a tree T, v has f fanin nodes $v_1, v_2, ..., v_f$ ($f \le K$). Let T_i denote the subtree in T rooted at v_i ($1 \le i \le f$). Clearly, any cut of size K in T induces a K_i-cut of T_i, with $\sum_i K_i = K$, and vice versa. Let $C_T(K)$ denote the set of cuts of size K in T, and define $C_T(1) = \{ v \}$, where v is the root of T. Then, we have (for $K > 1$)

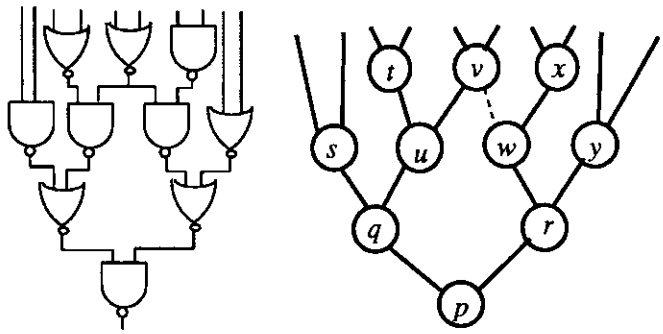
$$C_T(K) = \bigcup_{\sum_{i=1}^{f} K_i = K,\ K_i \ge 1} (C_{T_1}(K_1) * C_{T_2}(K_2) * \cdots * C_{T_f}(K_f)), \tag{1}$$

where for two sets of cuts A and B, $A * B$ is defined to be $\{c_1 * c_2 \mid c_1 \in A,\ c_2 \in B\}$.

Based on the recursive equation (1), we can generate all K-feasible cuts of a tree. Note that in this case, the number of cuts generated according to this equation is bounded by a constant that depends only on K and is independent of the size of $MFFC_v$.

5.3.2. Cut Generation for Non-Trees

If $MFFC_v$ is not a tree, We first construct a spanning tree T rooted at v, and then carry out the recursion on the spanning tree. Again, we assume that node v has f fanin nodes $v_1, v_2, ..., v_f$ ($f \le K$), and let T_i denote the subtree in T rooted at v_i ($1 \le i \le f$). However, a simple combination of the cuts in $T_1, T_2, \ldots, T_f$ does not always give a cut of $MFFC_v$. In Figure 7, The MFFC in (a) has a spanning tree shown in (b) where the dashed edge is not in the spanning tree. A combination of a cut su in the left subtree with a cut xy in the right subtree does not form a cut in the MFFC, since the edge $<v, w>$ provides a path connecting the root p to nodes outside of the MFFC. On the other hand, the cut $suvxy$ of the MFFC cannot be generated from the combinations of the cuts in the two subtrees, since suv is not a cut of the left subtree.



(a) (b)

Figure 7 Complication in cut generation.

The problem occurs because of the existence of the edges not in the spanning tree (called *non-tree* edges). If a non-tree edge $<u_i, u_j>$ crosses two subtrees T_i and T_j of the spanning tree T, we call u_i an *escape* node of T_i and u_j an *entrance* node of T_j. False cuts can be easily eliminated by examining the entrance nodes. In order to generate the cuts that are not combinations of the cuts of the subtrees, we generalize the concept of a cut. A *generalized* cut in a subtree of the spanning tree of an MFFC is a combination of a cut with some escape nodes. In Figure 7, *suv* is a generalized cut of the left subtree.

it is not difficult to show that the generalized cuts of tree T can be generated from the generalized cuts of its subtrees $T_1, T_2, \ldots, T_f$. Let $C_T(K)$ denote the set of generalized cuts of size K in tree T, and $E_T(K)$ denote the set of all the combinations of K escape nodes in T. Then, we have

$$C_T(K) \subseteq \Big[\bigcup_{\sum_{i=1}^{f} K_i = K} (C_{T_1}(K_1) * C_{T_2}(K_2) * \cdots * C_{T_f}(K_f)) \Big] \cup \big[\{ v \} * E_v(K-1) \big]. \tag{2}$$

Note that the last term $\{ v \} * E_v(K-1)$ represents the generalized cuts that cannot be generated from the combinations of the generalized cuts in the subtrees.

In fact, the set of all the combinations of K escape nodes, $E_T(K)$, can be recursively computed as well according to the following relation

$$E_T(K) \subseteq \bigcup_{\sum_{i=1}^{f} K_i = K} (E^*_{T_1}(K_1) * E^*_{T_2}(K_2) * \cdots * (E^*_{T_f}(K_f)), \tag{3}$$

where

$$E^*_{T_i}(K_i) = \begin{cases} E_{T_i}(K_i), & \text{if } v_i \text{ is not an escape node in } T \\ E_{T_i}(K_i) \cup [\ \{v_i\} * E_{T_i}(K_i - 1)\ ], & \text{if } v_i \text{ is an escape node in } T \end{cases} \quad (4)$$

Based on recursive equations (2), (3), and (4), we can compute all the generalized cuts of size no more than K in the spanning tree T of $MFFC_v$ efficiently, which include all the K-feasible cuts in $MFFC_v$.

Cut generation for general networks is more costly than for trees due to the existence of the escape nodes. Our experimental results showed that in practice, the above recursion (2) often quickly reaches the point where the subtree does not contain any escape node. In this case, the normal tree cuts generation algorithm is applied, which generates only a constant number of cuts (assuming that K is a constant). Our cut generation algorithm is much more efficient than the straight forward enumeration of K-node combinations. For $K = 5$, the number of all possible K-node combinations is $O(n^5)$, while our experimental results showed that on average, the total number of cuts generated by our algorithms is much smaller than n^3, where n is the number of nodes in an MFFC.

5.4. The Optimal DF-Mapping Algorithm

First, we show that we can collapse every K-feasible MFFC into its root prior to the mapping, without affecting the optimality of the subsequent DF-mapping.

Theorem 2 There exists an optimal DF-mapping solution in which every K-feasible MFFC is contained in a K-LUT.

Proof Let $MFFC_v$ be a K-feasible MFFC that is not contained in any K-LUT in an DF-mapping solution S. We will show that we can transform S into another solution S' that is at least as good as S, such that a K-LUT in S' contains $MFFC_v$.

Let LUT_u be the K-LUT in S that covers v. If $u = v$, we construct S' by replacing LUT_u with $LUT'_u = MFFC_v$ and eliminating all the LUTs implementing nodes in $MFFC_v$. Otherwise, let $W = \{w \mid w \in MFFC_v, w \notin LUT_u, output(w) \subseteq LUT_u\}$. Clearly, $|W| \geq 1$. Let $V = MFFC_v \cap LUT_u$, and $U = LUT_u - V$. Since v is the only node in $MFFC_v$ that may have fanout to U, $|input(U)| = |input(LUT_u)| - |W| + 1 \leq |input(LUT_u)|$. Therefore, U is also K-feasible. Since any K-LUT in S that implements a node in $MFFC_v$ must be contained by $MFFC_v$ (Lemma 3), we can transform S into S' by replacing LUT_u with $LUT'_u = U$, creating $LUT_v = MFFC_v$, and eliminating all the LUTs implementing the nodes in $MFFC_v$ (since

$| W | \geq 1$, there exists at least one). In both cases the transformation does not increase the number of K-LUTs, and $MFFC_v$ is contained in an LUT. Moreover, for any other K-feasible MFFC, if it is contained in a K-LUT in S, it is also contained in a K-LUT in S'. □

According to this theorem, we first collapse each K-feasible $MFFC_v$ into node v before the DF-mapping. According to our experimental results, this usually reduces the network size by 25% to 50% (when $K = 5$). Then, we use the dynamic programming approach to compute an optimal DF-mapping solution of $MFFC_v$ for each node v according to the topological order starting from the PI nodes. This order guarantees that when we compute the DF-mapping of $MFFC_v$, the optimal DF-mapping solutions of all the MFFCs inside $MFFC_v$ have been computed, so that we can evaluate the cost of each cut in $MFFC_v$ very easily. Finally, according to Theorem 1, we generate the optimal DF-mapping solution for the entire network starting from the PO nodes. Our area-optimal DF-mapping algorithm, called DF-Map, is summarized as follows.

```
algorithm DF-Map
  /* phase 0: collapse K-feasible MFFCs */
  for each node v do
    if MFFC_v is K-feasible then
      collapse MFFC_v into v;
  end-for;
  /* phase 1: compute optimal mapping for MFFCs */
  for each node v, in topological order starting from PI nodes, do
    compute MFFC_v;
    /* compute optimal DF-mapping for MFFC_v */
    mincost := ∞;
    for each K-feasible cut P =(X, X̄) of MFFC_v do
      decompose X into disjoint MFFCs MFFC_{v_i}, 1≤i≤m;
      cost(P) := 1 + Σ_{i=1}^{m} area(MFFC_{v_i});
      if mincost > cost(P) then
        LUT_v := X̄;  mincost := cost(P);
    end-for;
    area(MFFC_v) := mincost;
  end-for;
  /* phase 2: generate optimal mapping solution */
  L := list of PO nodes;  S := ∅;
  while L ≠ ∅ do
    remove a node v from L;
    S := S ∪ { optimal DF-mapping of MFFC_v };
    L := L ∪ input(MFFC_v);
  end-while;
  output S;
end-algorithm.
```

Based on the discussion in Sections 5.2 and 5.3, we have

Theorem 3 The DF-mapping problem for general Boolean networks in LUT-based FPGA designs can be solved optimally in polynomial time. □

6. Experimental Results

We have implemented the FlowMap-r algorithm on SUN Sparc workstations and tested it on a set of MCNC benchmark circuits. In order to make fair comparison with previous algorithms, we used the same initial networks as used by Chortle-crf/Chortle-d [FrRV91b], DAG-Map [ChCD92], and FlowMap [CoDi92]. These initial networks are synthesized using a MIS script [BrRS87] which performs technology independent optimization.

Table 1 shows the mapping solution sets computed by FlowMap-r. In general, larger networks have more room for area and depth trade-off. The area/depth trade-off curves for *rot*, *alu4*, and *des* are shown in Figure 8. For most circuits, as we increase the depth bound, the number of LUTs decreases considerably. The area reduction is usually more significant at the first a few steps of depth bound increase.

We also compared the the area- and depth-minimization solutions generated by FlowMap-r with those generated by some existing mapping algorithms. The data for these algorithms are quoted from [FrRV91b, CoDi92, MuSB91a][5]. Table 2 compares the area minimum solutions generated by FlowMap-r with those generated by area minimization mapping algorithms, including Chortle-crf and MIS-pga. Overall, the area-minimum solutions of FlowMap-r use 4% fewer LUTs and 15% fewer levels than Chortle-crf, and 2% fewer LUTs and 9% fewer levels than MIS-pga (on available data). Table 3 compares the depth-minimum solutions generated by FlowMap-r with those generated by depth minimization mapping algorithms, including FlowMap, MIS-pga(delay), and Chortle-d. Overall, the depth-optimal solutions of FlowMap-r use the same number of levels and 11% fewer LUTs than FlowMap, 8% fewer levels and 9% fewer LUTs than MIS-pga(delay), and 5% fewer levels and 41% fewer LUTs than Chortle-d. The improved version of MIS-pga program, MIS-pga(new) [MuSB91b], outperforms FlowMap-r in terms of area, but the depths of their solutions were not reported. It is important to point out that FlowMap-r is solely based on combinatorial optimization techniques, therefore runs faster than Boolean optimization based algorithms for large circuits. In our experiments, the largest

[4] There are a few benchmark circuits for which FlowMap-r cannot improve the startig solution of FlowMap by increasing the depth bounds, since for these circuits the solutions generated by FlowMap have optimal depth and near-optimal area. These circuits either are very small, or have tree-like structures.

[5] All the algorithms in the comparison, except MIS-pga(delay), are started with the same set of initial circuits that are initially used by Chortle-d. The data for MIS-pga on *des* is not available.

FlowMap-r Mapping Results for 5-LUT FPGAs						
Circuit	Opt. Depth d_{opt}	No. of 5-LUTs For Different Depths				
		d_{opt}+0	d_{opt}+1	d_{opt}+2	d_{opt}+3	d_{opt}+4
5xp1	3	23	22	-	-	-
C499	5	151	130	-	-	-
C880	8	211	195	179	172	-
alu2	8	148	140	133	-	125
alu4	10	245	244	240	231	223
apex6	4	232	221	220	-	-
apex7	4	80	76	-	-	-
count	4	73	57	-	-	-
des	5	1087	1003	987	969	934
duke2	4	187	172	161	151	-
rd84	4	43	42	38	-	-
rot	6	243	218	213	210	-

Table 1 Mapping solutions of FlowMap-r.[4]

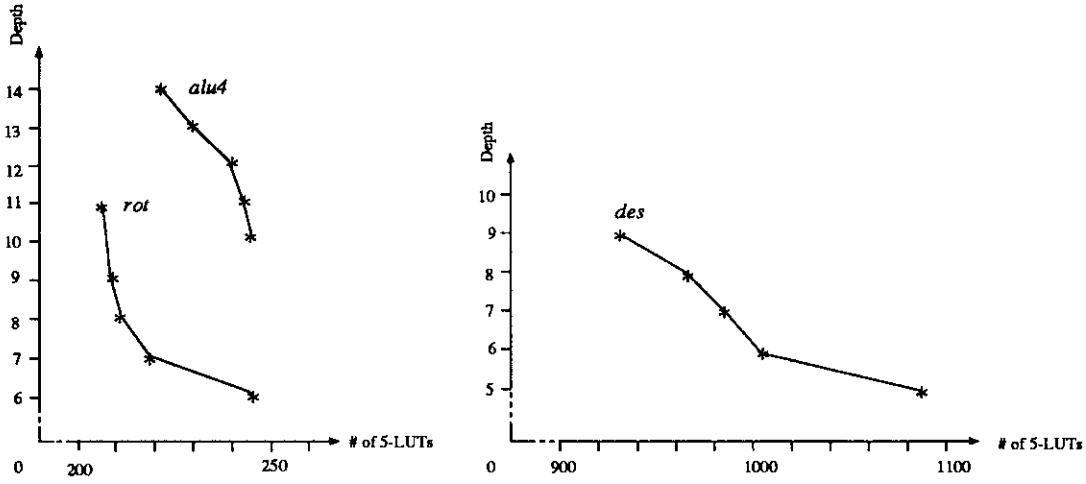


Figure 8 Area/depth trade-off in FlowMap-r (K=5).

5-LUT Mapping Result Comparison: FlowMap-r vs. Area Minimization Algorithms						
Circuit	**FlowMap-r**		**Chortle-crf**		**Mis-pga**	
	LUTs	**Depth**	**LUTs**	**Depth**	**LUTs**	**Depth**
5xp1	22	4	27	4	26	4
9sym	61	5	65	8	65	8
9symml	58	5	62	7	65	7
C499	130	6	141	8	123	7
C880	172	11	172	13	172	11
alu2	125	12	128	13	127	15
alu4	223	14	231	17	234	16
apex6	220	6	235	6	221	6
apex7	76	5	78	6	72	5
count	57	5	58	5	59	5
des	934	9	981	10	-	-
duke2	151	7	152	7	161	7
misex1	15	2	18	4	16	3
rd84	38	6	41	7	40	6
rot	209	11	214	11	203	11
vg2	38	4	39	5	37	5
z4ml	13	3	13	4	10	3
total	2542	115	2655	135	-	-

Table 2 Comparison with Chortle-crf and MIS-pga.

circuit *des* is mapped by FlowMap-r within 5 minutes of CPU time on a SUN Sparc IPC. Moreover, FlowMap-r produces a set of mapping solutions, each of them satisfies an explicitly assigned depth bound. Therefore, FlowMap-r gives designer more choices.

Finally, we have tested the effectiveness of the post-processing steps that are performed after DF-Map to further minimize the area of the mapping solution by necessary node duplications. Overall, the reduction on the number of LUTs is 5% to 10%. Considering the fact that the percentage of multi-fanout nodes is much larger than this, it further justified the assumption that an area-optimal mapping solution should not have large number of node duplications.

7. Conclusion

In this paper we have presented a technology mapping algorithm for LUT-based FPGA designs that is able to generate a set of mapping solutions with smooth area and depth trade-off. As part of the algorithm, we have developed an efficient method to compute an optimal mapping solution without node duplication for a general Boolean network, which is used for area

5-LUT Mapping Result Comparison: FlowMap-r vs. Depth Minimization Algorithms								
Circuit	**FlowMap-r**		**FlowMap**		**Mis-pga(delay)**		**Chortle-d**	
	LUTs	**Depth**	**LUTs**	**Depth**	**LUTs**	**Depth**	**LUTs**	**Depth**
5xp1	23	3	25	3	21	2	26	3
9sym	61	5	61	5	7	3	63	5
9symml	58	5	58	5	7	3	59	5
C499	151	5	154	5	199	8	382	6
C880	211	8	232	8	259	9	329	8
alu2	148	8	162	8	122	6	227	9
alu4	245	10	268	10	259	11	500	10
apex6	232	4	257	4	274	5	308	4
apex7	80	4	89	4	95	4	108	4
count	73	4	76	3	81	4	91	4
des	1087	5	1308	5	1397	11	2086	6
duke2	187	4	187	4	164	6	241	4
misex1	15	2	15	2	17	2	19	2
rd84	43	4	43	4	13	3	61	4
rot	243	6	268	6	322	7	326	6
vg2	38	4	45	4	39	4	55	4
z4ml	13	3	13	3	10	2	25	3
total	2908	83	3261	83	3182	90	4906	87

Table 3 Comparison with FlowMap, MIS-pga(delay), and Chortle-d.

minimization in our algorithm. The concept of a *maximum fanout free cone* plays an important role in our optimal duplication-free mapping algorithm, and it may finds applications to other logic synthesis problems as well. The solution set generated by our algorithm outperforms the solutions by many existing algorithms in terms of both area and depth. Although the *unit delay* model is used when describing the algorithm, we can generalize the algorithm to the case where an arbitrary delay is assigned to a net (for example, we can also handle the *nominal delay* model proposed by [ScCK91]). Due to the length restriction, this generalization is not presented in this paper.

During depth relaxation, we use only structural information to decompose the LUTs. It is also possible to use Boolean optimization techniques to re-synthesize the LUT network locally to explore more possibilities, at the expense of longer computation time.

The area-optimal mapping problem with node duplication for LUT-base FPGA designs remains an open problem. We are currently studying the problem of area-optimal mapping with bounded node duplications.

Acknowledgment

We thank Dr. K.C. Chen and Dr. Bryan Preas for their helpful discussions. We thank Bob Francis and Rajeev Murgai for their assistance in our comparative study. This research is partially supported by a grant from Xilinx Inc. under the State of California MICRO program No.92-030 and a grant from Fujitsu America Inc..

References

[BhHi92] 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.

[BrRS87] Brayton, R. K., R. Rudell, and A. L. Sangiovanni-Vincentelli, "MIS: A Multiple-Level Logic Optimization," *IEEE Transactions on CAD*, pp. 1062-1081, November 1987.

[ChCD92] 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*, Sep. 1992.

[CoDi92] Cong, J. and Y. Ding, "An Optimal Technology Mapping Algorithm fo Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Int'l Conf. on Computer Aided Design*, Nov. 1992.

[CoKT92] 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.

[FrRV91a] Francis, R. J., J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proceedings 28th ACM/IEEE Design Automation Conference*, pp. 613-619, 1991.

[FrRV91b] Francis, R. J., J. Rose, and Z. Vranesic, "Technology Mapping for Delay Optimization of Lookup Table-Based FPGAs," *MCNC Logic Synthesis Workshop*, 1991.

[Hi91] Hill, D., "A CAD System for the Design of Field Programmable Gate Arrays," *Proc. ACM/IEEE Design Automation Conference*, pp. 187-192, 1991.

[Ka91a] Karplus, K., "Xmap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays," *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 240-243,

1991.

[Mu90] Murgai, R., et al, "Logic Synthesis Algorithms for Programmable Gate Arrays," *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 620-625, 1990.

[MuSB91a] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," *Proc. Int'l Conf. Computer-Aided Design*, pp. 572-575, Nov., 1991.

[MuSB91b] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures ," *Proc. Int'l Conf. Computer-Aided Design*, pp. 564-567, Nov., 1991.

[SaTh92] 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.

[ScCK91] 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.

[ScKC92] 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*, Oct. 1992.

[Wo91] 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, 1991.

[Xi92] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, San Jose (1992).