Papers

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

Jason Cong, *Member, IEEE* and Yuzheng Ding

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 sequence of depth relaxation operations and area-minimizing mapping procedures to produce a set of mapping solutions for a given design with smooth area and depth trade-off. 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 K-bounded 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 most existing mapping algorithms in terms of both area and depth minimization.**

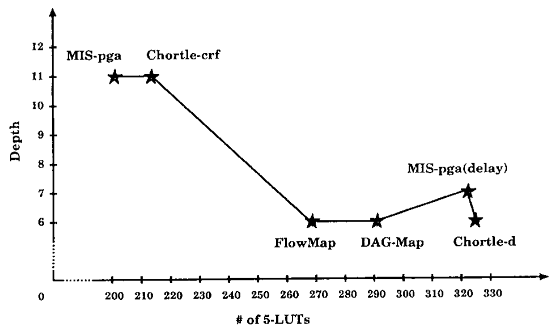


Fig. 1. Mapping solutions of various algorithms for *rot*($K = 5$).

I. 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 [11], [19], 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 transform a general Boolean network into a functionally equivalent K-LUT network.

Previous technology mapping algorithms for LUT-based FPGA designs can be roughly divided into three categories according to their optimization objectives. The algorithms in the first category emphasize on minimizing the number of LUT's in the mapping solution. These algorithms include Chortle-crf [9], MIS-pga [13], [15], XMap [12], VisMap [18], and TechMap [16]. The algorithms in the second category emphasize on minimizing the delay of the mapping solution. These algorithms include Chortle-d [10], MIS-pga(delay) [14], TechMap-L [16], DAG-Map [3], and FlowMap [4]. The algorithms in the third category, including RMap [17] and the algorithm reported in [1], 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.

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, Fig. 1 compares the 5-LUT mapping results by several existing algorithms on one of the MCNC benchmark circuits named *rot*. The depth and the number of LUT's of these solutions vary significantly. In general, the area-minimized solutions have much larger depth, while delay-minimized solutions use much more LUT's. In fact, we can construct an example circuit on which the depth-optimal solution has much larger area than the area-optimal solution, while the area-optimal solution has much larger depth than the depth-optimal solution (see Fig. 2). However, it is very likely that in practice the best design does not come from either of these two extremal solutions. 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.

Manuscript received April 13, 1993; revised December 1, 1993. This work was supported in part by a Grant from Xilinx Inc., under the State of California MICRO Program 92-030, a Grant from Fujitsu America Inc. and the National Science Foundation Young Investigator Award under Grant MIP-9357582.

The authors are with the Department of Computer Science, University of California, Los Angeles, CA 90024 USA.

IEEE Log Number 9400591.

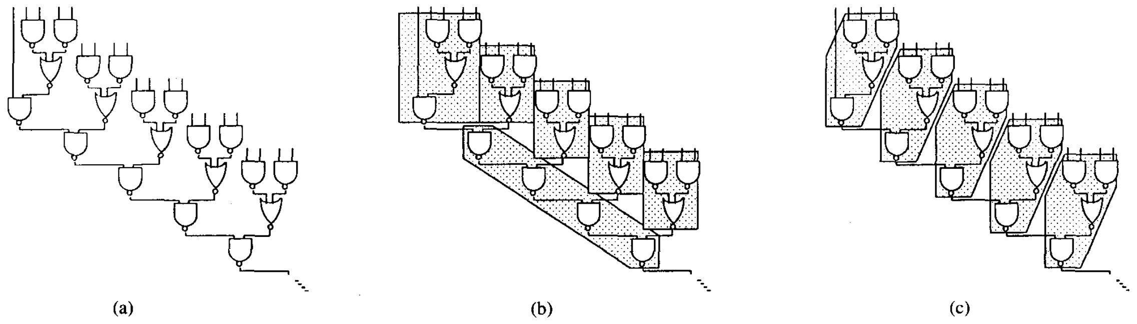


Fig. 2. Depth-optimal versus area-optimal mapping. (a) Part of a circuit consisting of $4n+1$ identical 4-gate segments. The depth of the circuit is $4n+3$. (b) The depth-optimal 5-LUT mapping soluton of depth $n+1$, using $5n+1$ LUT's. (c) The area-optimal 5-LUT mapping solution of depth $4n+1$, using $4n+1$ LUT's.

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 a given 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 the system 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 [4]), we perform a sequence 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, we are able to produce a *set* of mapping solutions with smooth area and depth trade-off for a given design. 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. In fact, it was shown very recently that area-optimal mapping with node duplication for K-bounded network is NP-Hard [8].

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 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 II formulates the problem and introduces several concepts and definitions. Section III presents an overview of our algorithm. In Sections IV and V, the details of the two phases of our algorithm, i.e., depth relaxation and area minimization, are discussed. Section VI presents the experimental results. Conclusions and future extensions are presented in Section VII.

II. 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 PI's) 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) *of* 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 nonroot nodes in C_v 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

[1] If a network is not K-bounded, there are a few algorithms to transform it into a K-bounded network. For example, the DMIG algorithm in [3] 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.

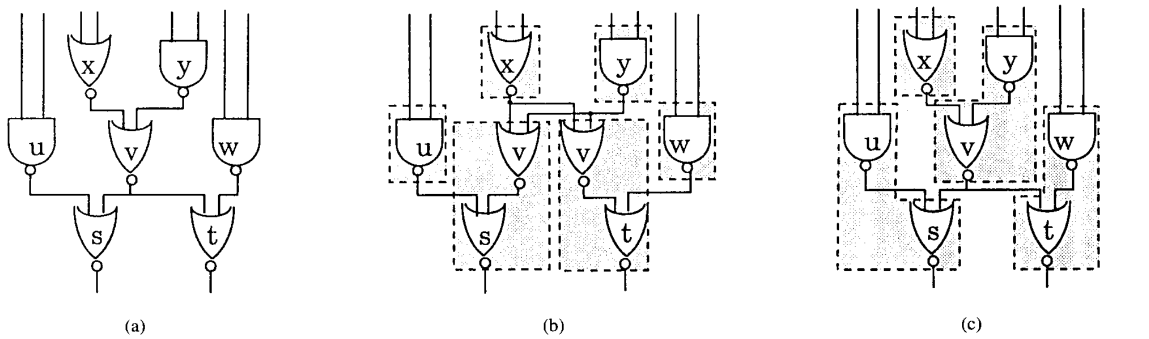


Fig. 3. Technology mapping for LUT-base FPGA ($K = 3$). (a) Original network; (b) mapping with node duplication; (c) mapping without node duplication.

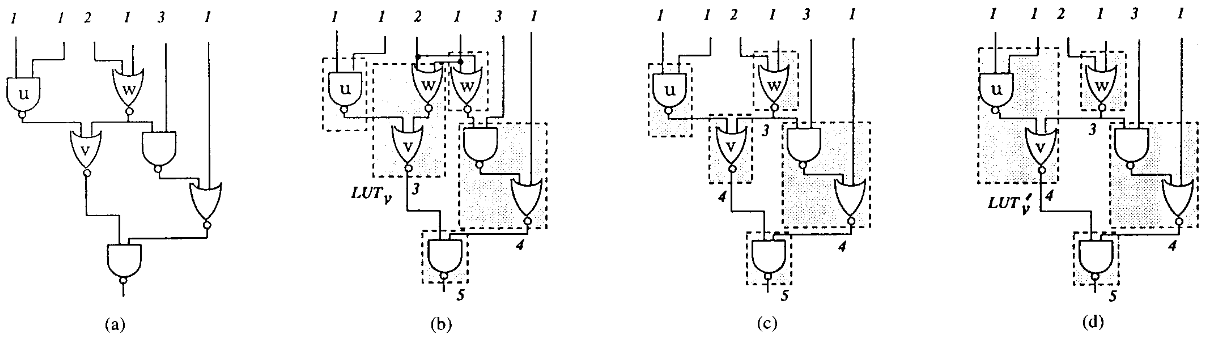


Fig. 4. Depth relaxation for area reduction ($K = 3$). The numbers indicate node levels. (a) Original network; (b) solution of FlowMap; (c) after depth relaxation; (d) after re-mapping for area minimization.

with K-feasible FFC's (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). Fig. 3 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.*

III. Basic Operations and Outline of the Algorithm

In this section, we first 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. We start with a brief description of the FlowMap algorithm, which will be used to compute a depth-optimal mapping solution as our starting point.

A. The FlowMap Algorithm

FlowMap [4] 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.

B. Effect of Depth Relaxation

Insisting minimum depth for every node, including the noncritical ones, may lead to inefficient use of LUT's. Fig. 4(a) shows a Boolean network. The mapping solution by FlowMap is shown in Fig. 4(b). Another solution is shown in Fig. 4(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 repack LUT$_u$ into LUT$_v$. Since the decomposition increases the depth of 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 IV.

C. Effect of Node Duplication

Node duplication is performed when we use an LUT to cover a K-feasible cone C which has a nonroot node with a fanout node outside of C [see Fig. 3(b)]. In general, node duplication is very important to depth optimization, as duplication usually increases the parallelism in the circuit. Without node duplication, we may have to implement many multifanout nodes explicitly with LUT's, 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 multifanout 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 LUT's, and it may use certain input capacity of each LUT. Therefore, excessive node duplication will very likely result in large number of LUT's. Fig. 3 shows a typical example where node duplication reduces the depth but increases the area of the mapping solution.

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 noncritical nodes, until all the remaining duplications are necessary to guarantee 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.

D. 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 sequence of depth relaxation operations to produce an intermediate network suitable for subsequent area minimization. In this phase, the depth of certain noncritical nodes will be increased, but the depth bound on the intermediate network is maintained. 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 *duplication-free* mapping solution, i.e., a mapping solution without node duplication. The details of DF-Map will be presented in Section V. Then, we carry out two post-processing procedures which allow necessary node duplications for further area minimization. The two procedures are MP-Pack, a multifanout predecessor packing procedure from the DAG-Map package [3], and Flow-Pack, a flow-based area minimization procedure from the FlowMap package [4].

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 nonzero slacks do
      select a node with nonzero slack;
      apply a depth relaxation operation
        to decompose the node;
      recompute slacks;
    end-while;
    /* phase 2: area minimization */
    call DF-Map to perform area-optimal
      duplication-free mapping;
    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.
```

IV. Depth Relaxation

Given a noncritical LUT LUT_v rooted at a node v and some node $w \in \text{LUT}_v$, the *depth relaxation operation* applied on LUT_v and node w decomposes LUT_v into LUT'_v and LUT_w, so that LUT_w becomes a fanin of LUT'_v. Node w is called the *breaking node*. If LUT_w already exists in the mapping solution (i.e., w is a duplicated node in LUT_v), the depth relaxation simply replaces LUT_v with LUT'_v and let LUT_w be a fanin of LUT'_v, as in Fig. 4(c). Otherwise, LUT_w will be created explicitly.

Due to the depth bound, not all the LUT's can be decomposed, and the depth relaxation operation may lead to different results when applied on different LUT's, or in different ways on the same LUT. The problem of finding an *optimal* decomposition is difficult due to the prohibitively large number of possible configurations. We use a greedy heuristic approach that always applies the "most promising" depth relaxation operations first. For every LUT LUT_v with nonzero slack and every possible breaking node $w \in \text{LUT}_v$, we estimate the *potential* area reduction by decomposing LUT_v into LUT'_v and LUT_w, and choose the best one for decomposition. The estimation may be based on the reduction of input sizes of the decomposed and adjacent LUT's, the elimination or potential elimination of LUT's due to merging duplications and eligible repacking of adjacent LUT's, and the effect on the slack consumption of the other LUT's. In particular, the heuristic evaluation function we used is of the form

$$p(\text{LUT}_v, w) = (\alpha R_{\text{ex}} + \beta R_{\text{im}} + \gamma)/(R_{\text{slk}} + 1),$$

where R_{ex} is the *explicit reduction* in the number of LUT's due to the elimination of LUT's whose output signal are no longer needed; R_{im} is the *implicit reduction* in the number of LUT's, estimated by local repacking and by considering further decomposition of adjacent LUT's; and R_{slk} is the number of PO's whose slacks are decreased by the decomposition. Note that in the worst case R_{ex} is -1 since the decomposition first creates a new LUT_w. However, if LUT_w already exists in the current mapping solution, we always have $R_{\text{ex}} \geq 0$. In general, we choose $\alpha > \beta > 0$. In our experiments, we set $\alpha = 4\beta$.

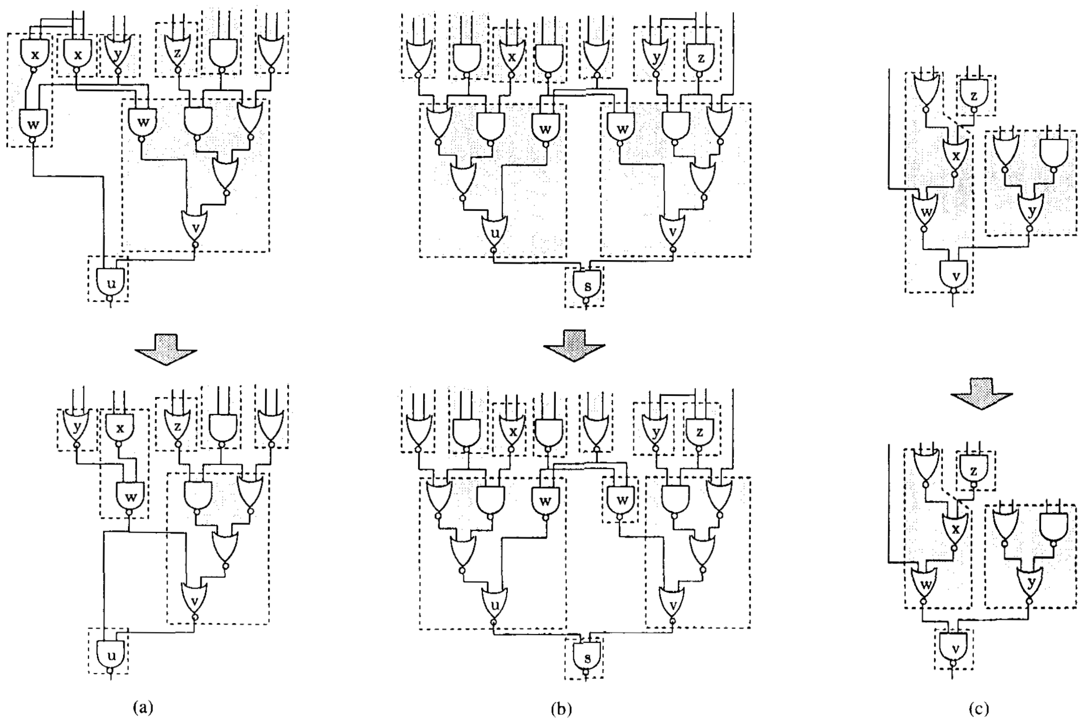


Fig. 5. Three types of depth relaxation operations (assume $K = 5$ and LUT_v has nonzero slack).

The impact of input size reduction is implicitly considered in R_{im}, but can also be explicitly included in the function.

Fig. 5 illustrates three types of depth relaxation, which are considered in our algorithm.

In Fig. 5(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 using w as the breaking node, 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 LUT's. 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 LUT's of w, which may either enable further packing of LUT_w with such a fanin LUT (in this example, LUT_y), or elimination of a redundant duplication of the fanin node (in this example, node x).

In Fig. 5(b), the two duplications of node w are in LUT_u and LUT_v. Since LUT_w needs to be explicitly created when we choose w as the breaking node, 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. 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 Fig. 5(c), LUT_v contains a node w which has single fanout. However, using w as the breaking node to decompose LUT_v 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 beneficial.

Note that for a given LUT_v, the choice of the breaking node is greedy. That is, we always choose $w \in \mathrm{LUT}_v$ for decomposition of LUT_v such that $p(\mathrm{LUT}_v, w)$ is maximum. For example, in Fig. 5(c), since choosing w as the breaking node leads to maximum area reduction, the depicted depth relaxation is applied.

After a depth relaxation operation, the slacks and the potentials of the *affected* nodes are recomputed, and the depth relaxation is performed again on the next node with nonzero slack and maximum potential. This 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.

We now briefly analyze the complexity of depth relaxation. Assuming that the number of nodes in the original network N is n. Then, we can apply depth relaxation at most n times. After each depth relaxation operation, we update the slacks of some or all of the nodes in the LUT network and some or all of the $p(\mathrm{LUT}_v, w)$ values for $w \in N$ which is a possible breaking node of LUT_v. Slack computation for the entire network takes $O(Kn)$ time. For each node $w \in N$, there are at most $|output(w)|$ LUT's that may contain w as a breaking node, where $output(w)$ is the set of fanout nodes of w in N. Since N is K-bounded, the total number of $p(\mathrm{LUT}_v, w)$ values that we need to evaluate is bounded by

$$\sum_{w \in N} |output(w)| = \sum_{w \in N} |input(w)| \leq Kn.$$

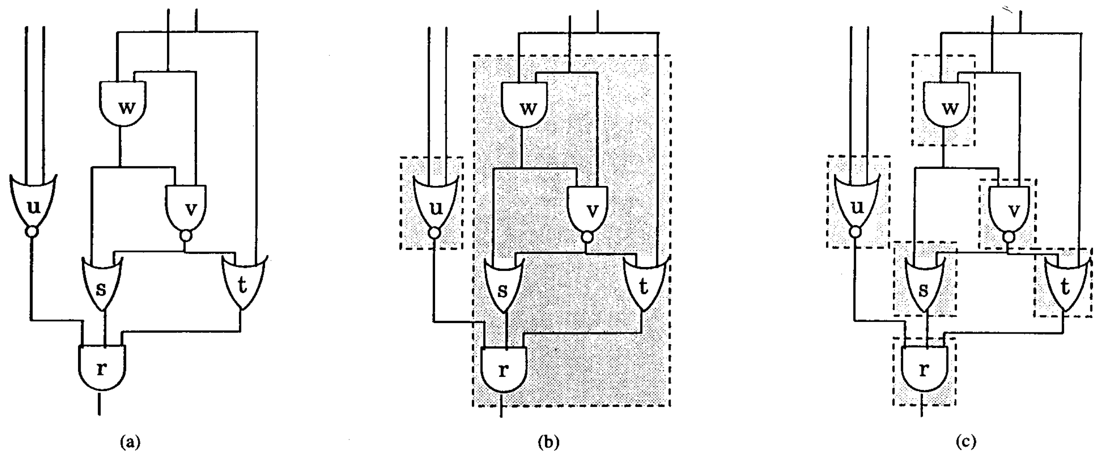


Fig. 6. Duplication-free mapping versus tree-based mapping ($K = 3$). (a) Orginal network; (b) duplication-free mapping; (c) tree-based mapping.

Recall that $p(\mathrm{LUT}_v, w)$ contains three components $R_{\mathrm{ex}}, R_{\mathrm{im}}$, and R_{slk}. Each R_{ex} and R_{im} can be computed in constant time since they depend on only local information. Let RPO_v denote the set of PO's whose slacks will decrease if the depth of LUT_v increases by one. Let F_v denote the set of fanouts of LUT_v such that the depth of each node in F_v is the depth of LUT_v plus one. Define

$$\mathrm{RPO}_v = \begin{cases} \{v\} & \text{if } v \text{ is a PO} \\ \cup_{w \in F_v} \mathrm{RPO}_w & \text{otherwise} \end{cases}$$

It is not difficult to show that when computing $p(\mathrm{LUT}_v, w)$, either $R_{\mathrm{slk}} = 0$ (if the decomposition does not increase the depth of LUT_v), or $R_{\mathrm{slk}} = |\mathrm{RPO}_v|$. Since the total number of fanouts is bounded by Kn, the computation of RPO_v for all LUT_v takes no more than $O(Knr)$ time, where r is the number of PO's. Therefore, all of the $p(\mathrm{LUT}_v, w)$ values in the network can be computed in $O(Knr)$ time. In summary, updating the slacks and $p(\mathrm{LUT}_v, w)$ after each depth relaxation takes no more than $O(Knr)$ time, Therefore, the complexity of the depth relaxation is bounded by $O(Kn^2r)$. In practice, the computational complexity is much lower since we update only the slacks and potential function values of the affected nodes.

V. Area Optimal Duplication-Free Mapping

In this section we present a polynomial time algorithm for area-optimal duplication-free mapping (DF-mapping) for general K-bounded 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. Fig. 6 shows a simple example where the optimal DF-mapping uses 2 LUT's, while the optimal tree-based mapping uses 6 LUT's. Our algorithm is based on an important concept called the *maximum fanout free cone*.

A. 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,

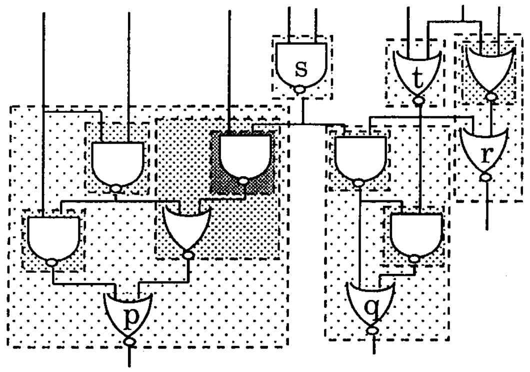


Fig. 7. Maximum fanout free cones.

if *output*$(w) \subseteq \mathrm{MFFC}_v$, then $w \in \mathrm{MFFC}_v$. Fig. 7 shows the MFFC of each node (the smallest shadowed area) in a network. Clearly, the MFFC of each node is unique, and any FFC of v is contained in MFFC_v. Moreover, MFFC has the following important properties.

Lemma 1: If $w \in \mathrm{MFFC}_v$, then $\mathrm{MFFC}_w \subseteq \mathrm{MFFC}_v$.

Proof: For any node $u \in \mathrm{MFFC}_w$, 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*$(w') \not\subseteq \mathrm{MFFC}_w$, which contradicts the assumption that $w' \in \mathrm{MFFC}_w$. Therefore, every path from u to a PO node must pass w. Similarly, since $w \in \mathrm{MFFC}_v$, 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*$(u) \subseteq \mathrm{MFFC}_v$, i.e. $u \in \mathrm{MFFC}_v$. Therefore, $\mathrm{MFFC}_w \subseteq \mathrm{MFFC}_v$. □

Lemma 2: Two MFFC's are either disjoint or one must contain another.

Proof: If MFFC_v and MFFC_w are not disjoint, let $u \in \mathrm{MFFC}_v \cap \mathrm{MFFC}_w$. 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 $w \in \mathrm{MFFC}_v$, and according to Lemma 1, $\mathrm{MFFC}_w \subseteq \mathrm{MFFC}_v$. □

Lemma 3: If LUT_w is in a DF-mapping solution S, then $w \in \mathrm{MFFC}_v$ implies $\mathrm{LUT}_w \subseteq \mathrm{MFFC}_v$ for any nodes v and w.

Proof: Since there is no node duplication, it is clear that LUT_w implements an FFC rooted at w. Therefore, $\mathrm{LUT}_w \subseteq \mathrm{MFFC}_w$. Since $w \in \mathrm{MFFC}_v$, according to Lemma 1 we know $\mathrm{MFFC}_w \subseteq \mathrm{MFFC}_v$. Thus, $\mathrm{LUT}_w \subseteq \mathrm{MFFC}_v$. □

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

B. MFFC Partitioning of General Network

First, we show that a general Boolean network can be decomposed into a set of disjoint MFFC's 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, since v is a PO, $v \notin \mathrm{MFFC}_u$. If $u \in \mathrm{MFFC}_v$, according to Lemma 3, $\mathrm{LUT}_u \subseteq \mathrm{MFFC}_v$. If $u \notin \mathrm{MFFC}_v$, according to Lemma 2, $\mathrm{MFFC}_u \cap \mathrm{MFFC}_v = \emptyset$, which implies $\mathrm{LUT}_u \cap \mathrm{MFFC}_v = \emptyset$. Therefore, any LUT in S is either contained in MFFC_v or disjoint with MFFC_v. As a result, S induces a mapping solution S_v in MFFC_v. Moreover, S_v must be optimal, for otherwise, by improving S_v, 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 MFFC's so that we can compute the optimal DF-mapping for each MFFC independently to obtain an optimal DF-mapping solution of N. In Fig. 7, the MFFC's 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.

C. Optimal DF-Mapping for MFFC's

Assume that we want to compute an area-optimal DF-mapping solution of MFFC_v (it will be clear from the following discussion that depth-optimal DF-mapping solution can be computed in a similar way). First, we introduce some basic concepts about *cuts* in MFFC_v.

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

For each K-feasible cut $P = (X, \overline{X})$ of MFFC_v, we can cover $\overline{X}$ with a K-LUT LUT_v^P, and partition $X = \mathrm{MFFC}_v - \overline{X}$ into a set of disjoint MFFC's $\mathrm{MFFC}_{v_1^p}, \mathrm{MFFC}_{v_2^p}, \cdots, \mathrm{MFFC}_{v_m^p}$. Then, we recursively compute the area-optimal DF-mapping of each $\mathrm{MFFC}_{v_i^p}$ $(1 \le i \le m)$. The *cost* of the cut P is defined to be $cost(P) = 1 + \sum_{i=1}^{m} area(\mathrm{MFFC}_{v_i^p})$, where $area(\mathrm{MFFC}_{v_i^p})$ is the area of the area-optimal DF-mapping of $\mathrm{MFFC}_{v_i^p}$. Clearly, $cost(P)$ gives the area of the best DF-mapping solution of MFFC_v if $\overline{X}$ 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 MFFC's of smaller sizes.

The same method can be applied to other optimization objectives. For example, if we want to compute a depth-optimal DF-mapping, we simply change the cost function to $cost(P) = 1 + max_{i=1}^{m} depth(\mathrm{MFFC}_{v_i^p})$, where $depth(\mathrm{MFFC}_{v_i^p})$ is the depth of the depth-optimal DF-mapping of $\mathrm{MFFC}_{v_i^p}$. (In fact, depth-optimal DF-mapping can be computed more efficiently. Using the FlowMap [4] approach, it will only take $O(Kmn)$ time to map a network of n nodes and m edges.)

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 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 $(X, \overline{X})$ by a string $v_1 v_2 \cdots v_m$, where $input(\overline{X}) = \{v_1, v_2, \cdots, v_m\}$. 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., $v_1 v_2 \cdots v_m * u_1 u_2 \cdots u_n = v_1 v_2 \cdots v_m u_1 u_2 \cdots u_n$. For convenience, we use ϕ to denote the *empty* string, i.e., $v * \phi = \phi * v = v$. Finally, for two sets of cuts A and B, $A * B$ is defined to be $\{c_1 * c_2 \mid c_1 \in A, c_2 \in B\}$ if $A \ne \emptyset$ and $B \ne \emptyset$, and $\emptyset$ otherwise.

Cut Generation for Trees Assume that MFFC_v is a tree T, v has f fanin nodes $v_1, v_2, \cdots, v_f (f \le K)$. 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 $K > 1$)

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

Based on the recursive equation (1), we can generate all K-cuts of a tree. Note that in this case, the number of cuts generated according to this equation is bounded by a constant depending only on K. In fact, it can be shown that this constant is bounded by the $(K-1)$th *Catalan number* [7], denoted c_{K-1}, where $c_K = \frac{1}{K+1}\binom{2k}{k}$. For $K = 5$, $c_{K-1} = 14$. Therefore, when MFFC_v is a tree, the total number of K-feasible cuts that we need to generate is bounded by $\sum_{i=0}^{K-1} c_i$, which is 23 for $K = 5$.

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 $v_1, v_2, \cdots, v_f (f \le K)$, and let T_i denote

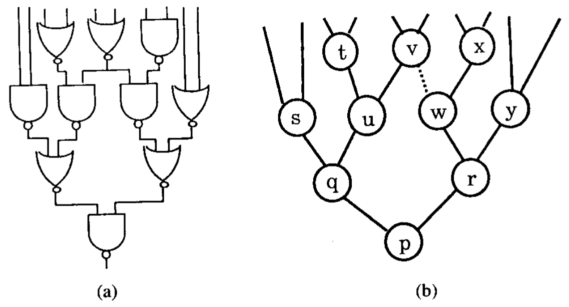


Fig. 8. Complication in cut generation.

the subtree in T rooted at v_i $(1 \leq i \leq f)$. However, a simple combination of the cuts in $T_1, T_2, \cdots, T_f$ does not always give a cut of MFFC$_v$. In Fig. 8, 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.

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 in T and u_j an *entrance* node of T_j in T. 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 Fig. 8, suv is a generalized cut of the left subtree by including the escape node v.

It is not difficult to show that the generalized cuts of tree T can be generated from the generalized cuts of its subtrees $T_1, T_2, \cdots, T_f$. Specifically, 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. Clearly, any generalized cut of size K that contains the root v of T is in the set $\{v\} * E_T(K-1)$, since v itself forms a cut in T. On the other hand, any generalized cut that does not contain v can be formed by combinations of the generalized cuts of the subtrees of T, since the nodes that form a (normal) cut will form a cut in each of the subtrees. Therefore, we have

$$C_T(K) \subseteq \left[\bigcup_{\sum_{i=1}^{f} K_i = K, K_i \geq 0} (C_{T_1}(K_1) * C_{T_2}(K_2) * \cdots * C_{T_f}(K_f)) \right] \cup [\{v\} * E_T(K-1)]. \tag{2}$$

Note that in this recursion, a subtree T_i may have a "cut" of size 0, if during the construction of the spanning tree, all the paths from the leaves of the spanning tree to the root of T_i have been cut off. In this case, $C_{T_i}(0) = \{\phi\}$.

Based on (2), $C_T(K)$ can be recursively computed if $E_T(K-1)$ is known. In general, the recursion stops when the subtree T_i is a single node, where we have

$$\begin{aligned} C_{T_i}(0) &= \begin{cases} \{\phi\} & \text{if } T_i \text{ is not a leaf of } T \\ \emptyset & \text{otherwise} \end{cases} \\ C_{T_i}(1) &= \{v_i\} \\ C_{T_i}(K) &= \emptyset \qquad \text{for } K > 1. \end{aligned} \tag{2$'$}$$

In fact, when we reach a subtree T_i that does not contain any escape or entrance nodes, the recursion (1) for tree can be used to simplify computation.

We shall now discuss the computation of $E_T(K)$. Note that $E_T(0) = \{\phi\}$ is always true. For convenience, we define $E_T(K) = \emptyset$ for $K < 0$. For $K > 0$, $E_T(K)$ can be recursively computed as well, according to the following relation:

$$E_T(K) \subseteq \bigcup_{\sum_{i=1}^{f} K_i = K, K_i \geq 0} \left(E^+_{T_1}(K_1) * E^+_{T_2}(K_2) * \cdots * (E^+_{T_f}(K_f) \right), \tag{3}$$

where

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

for $K_i > 0$, and $E^+_{T_i}(0) = \{\phi\}$.

In (3) and (4), $E_{T_i}(K_i)$ gives the set of the combinations of K escape nodes without considering the root v_i of T_i. $E^+_{T_i}(K_i)$ gives the set of the combinations of K escape nodes with possible inclusion of v_i when it is an escape node in T. The use of $E^+_{T_i}(K_i)$ simplifies the recursive relation in (3). The recursions in (3) and (4) stop when the subtree T_i does not contain any escape node other than its root v_i. In this case we have

$$\begin{aligned} E_{T_i}(0) &= \{\phi\} \\ E_{T_i}(K) &= \emptyset \quad \text{for } K > 0, \end{aligned} \tag{3$'$}$$

and

$$\begin{aligned} E^+_{T_i}(0) &= \{\phi\} \\ E^+_{T_i}(1) &= \begin{cases} \{v_i\} & \text{if } v_i \text{ is an escape node of } T \\ \emptyset & \text{otherwise} \end{cases} \\ E^+_{T_i}(K) &= \emptyset \qquad \text{for } K > 1. \end{aligned} \tag{4$'$}$$

Based on (2), (3), and (4), $C_T(K_i)$ and $E_T(K_i)$ $(1 \leq K_i \leq K)$ can be computed simultaneously for each subtree, in the topological order starting from the leaves in T. This yields an efficient way of generating all the generalized cuts of size no more than K in the spanning tree T of MFFC$_v$, 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. However, 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 small constant number of cuts. Our cut generation algorithm is much more efficient than the straight forward enumeration of all K-node combinations. For $K = 5$, the number of all possible K-node combinations is $\Theta(n^5)$, while

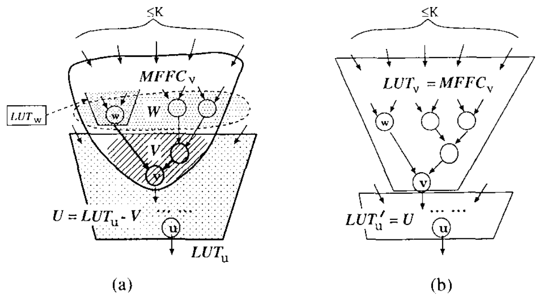


Fig. 9. Transformation of DF-mapping solutions. (a) In the original solution, MFFC_v is not contained in any LUT. (b) In the transformed solution, MFFC_v is contained in an LUT.

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.

D. 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 a 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. We consider two cases. i) If $u = v$, we construct S' by replacing LUT_u with $\mathrm{LUT}'_u = \mathrm{MFFC}_v$ and eliminating all the LUT's implementing nodes in MFFC_v. ii) If $u \neq v$, let $W = \{w | w \in \mathrm{MFFC}_v, w \notin \mathrm{LUT}_u, \mathit{output}(w) \subseteq \mathrm{LUT}_u\}$. Clearly, $|W| \geq 1$. Let $V = \mathrm{MFFC}_v \cap \mathrm{LUT}_u$ and $U = \mathrm{LUT}_u - V$. See Fig. 9(a) for illustration. Since v is the only node in MFFC_v that may have fanout to U, $|\mathit{input}(U)| = |\mathit{input}(\mathrm{LUT}_u)| - |W| + 1 \leq |\mathit{input}(\mathrm{LUT}_u)|$. 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 $\mathrm{LUT}'_u = U$, creating $\mathrm{LUT}_v = \mathrm{MFFC}_v$, and eliminating all the LUT's implementing the nodes in MFFC_v (since $|W| \geq 1$, there exists at least one such LUT). This is shown in Fig. 9(b).

In both cases, the transformation does not increase the number of K-LUT's, 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 remains contained in the corresponding K-LUT in S'. □

According to this theorem, we first collapse each K-feasible MFFC_v into node v before the DF-mapping. Our experimental results showed that 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 MFFC's 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 MFFC's */
  **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 MFFC's */
  **for** each node $v$, in topological order starting from PI nodes, **do**
      compute MFFC $_v$;
      /* compute optimal DF-mapping for MFFC $_v$ */
      mincost := $\infty$;
      **for** each $K$-feasible cut $P = (X, \overline{X})$ of MFFC $_v$ **do**
        decompose $X$ into disjoint MFFC's MFFC $_{v_i^P}$, $1 \leq i \leq m$;
        $cost(P) := 1 + \sum_{i=1}^{m} area$ ( MFFC $_{v_i^P}$);
        **if** mincost $>$ $cost(P)$ **then**
          LUT$_v$ := $\overline{X}$; mincost :=$cost(P)$;
      **end-for**;
      $area$( MFFC $_v$) := min cost;
  **end-for**;
  /* phase 2: generate optimal mapping solution */
  $L$ := list of PO nodes; $S := \emptyset$;
  **while** $L \neq \emptyset$ **do**
      remove a node $v$ from $L$;
      $S := S \cup \{$ optimal DF-mapping of MFFC$_v$ $\}$;
      $L := L \cup$ input(MFFC$_v$);
  **end-while**;
  **output** $S$;
**end-algorithm.**
```

Based on the discussion in Sections V-B and -C, we have

Theorem 3: The DF-mapping problem for general K-bounded Boolean networks in LUT-based FPGA designs can be solved optimally in polynomial time when K is a constant. □

E. Integration of Depth Relaxation and DF-Mapping

As described in Section III-D, the DF-Map algorithm is used in the second phase of FlowMap-r to perform area minimization. The initial network of DF-Map is generated using the FlowMap algorithm followed by depth relaxation.

Since every new LUT generated by DF-Map covers at least one node of the initial network, the depth of the DF-Map solution is no larger than the initial network to DF-Map. Since the depth bound is maintained during the depth relaxation, it is still maintained after DF-Map. Moreover, according to the results in [3], [4], the two post-processing procedures MP-Pack and Flow-Pack that are used in FlowMap-r for area minimization with node duplication, will not increase network depth either. Therefore, after the depth relaxation phase in FlowMap-r, the depth of the network will not increase during the area minimization. This guarantees that FlowMap-r can control the depth of the final solution successfully.

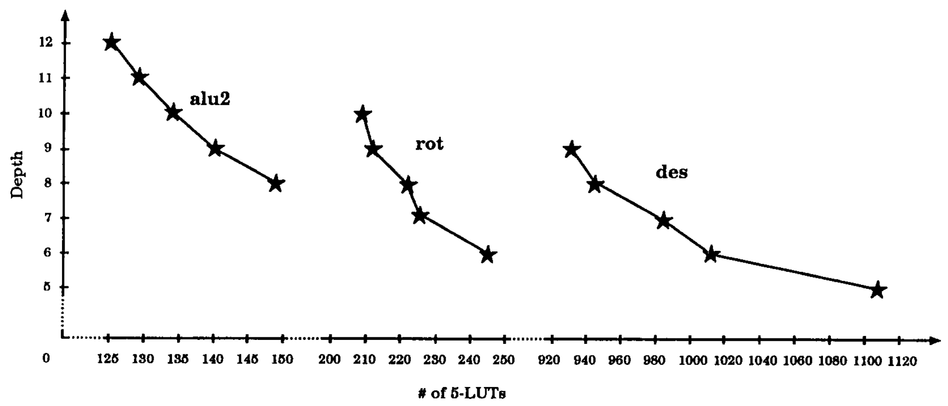


Fig. 10. Area/depth trade-off in FlowMap-r ($K = 5$).

The objective of the depth relaxation phase is to prepare an advantageous initial network for DF-Map. In Section IV, it can be seen that certain depth relaxation operations will alter the MFFC structure, as in the case of Fig. 5(a), while others will not, as in the case of Fig. 5(c). In general, the MFFC structure may be affected only when a duplication of a multiple fanout node is eliminated by the operation. Clearly, operations of the former type will have impact on the DF-Map solution since unnecessary duplications are removed. On the other hand, the operations of the latter type are also necessary, since further decomposition of a larger MFFC is an important part of DF-Map. As an example, Fig. 5(c) consists of a single MFFC which is not 5-feasible. Without depth relaxation, MFFC$_v$ consists of 3 nodes, and the optimal mapping is the existing mapping using 3 LUT's. After depth relaxation, MFFC$_v$ consists of 4 nodes, and the optimal mapping yields a solution of 2 LUT's.

VI. 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 [10], DAG-Map [3], and FlowMap [4]. These initial networks are synthesized using a MIS script [2] which performs technology independent optimization.

Table I shows the mapping solution sets computed by FlowMap-r. The time in this table is the CPU time used for the solution of the maximum depth relaxation shown in the table, recorded on a SUN SPARC IPC (14.8MIPS). In general, larger networks have more room for area and depth trade-off. The area/depth trade-off curves for *alu2, rot* and *des* are shown in Fig. 10. For most circuits, as we increase the depth bound, the number of LUT's decreases considerably.[1]

We also compared the area-minimum and depth-minimum solutions generated by FlowMap-r with those generated by several existing mapping algorithms. The data for these algorithms are quoted from [4], [10], [14][2]. Table II compares the area-minimum solutions generated by FlowMap-r with those generated by other mapping algorithms for area minimization, including Chortle-crf and MIS-pga. Overall, the area-minimum solutions of FlowMap-r use 5% fewer LUT's and 18% fewer levels than Chortle-crf, and 2% fewer LUT's and 14% fewer levels than MIS-pga (on available data). Table III compares the depth-minimum solutions generated by FlowMap-r with those generated by other mapping algorithms for depth minimization, including FlowMap, MIS-pga(delay), and Chortle-d. Overall, the depth-optimal solutions of FlowMap-r use the same number of levels and 10% fewer LUT's than FlowMap, 8% fewer levels and 9% fewer LUT's than MIS-pga(delay), and 5% fewer levels and 40% fewer LUT's than Chortle-d. The improved version of MIS-pga program, MIS-pga(new) [15], 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 circuit *des*, which contains 3263 two-input gates, is mapped by FlowMap-r in about 7 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, while other mapping methods produce only a single solution. Therefore, FlowMap-r gives designer more choices.

The result of the depth relaxation operations has significant impact on the effectiveness of the area minimization. We have used a greedy heuristic algorithm to perform depth relaxation. To test the effectiveness of the heuristic, we also implemented a depth relaxation algorithm that picks up noncritical nodes for decomposition in the topological order starting from the nodes adjacent to PI's. Compared with this method, for the same level of depth relaxation our heuristic always resulted in fewer LUT's in the final solution. For example, our heuristic

[1] There are a few benchmark circuits for which FlowMap-r cannot improve the starting 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.

[2] 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.

TABLE I
MAPPING SOLUTIONS OF FLOWMAP-r

FlowMap-r Mapping Results for 5-LUT FPGAs								
Circuit	No. of Nodes before Mapping	Optimal Depth d_{opt}	No. of 5-LUTs For Different Depths					CPU Time (sec.)
			$d_{opt}+0$	$d_{opt}+1$	$d_{opt}+2$	$d_{opt}+3$	$d_{opt}+4$	
rd84	141	4	47	39	37	-	-	11.8
count	216	3	76	58	-	-	-	10.1
apex7	247	4	83	78	-	-	-	12.9
duke2	392	4	188	167	148	-	-	42.2
alu2	393	8	149	140	134	129	125	51.6
C880	548	8	206	195	177	170	-	73.0
rot	647	6	246	225	222	212	209	80.7
C499	658	5	134	129	-	-	-	87.9
alu4	726	10	253	243	237	223	-	106.4
apex6	779	4	232	222	220	218	-	123.7
des	3263	5	1109	1013	983	944	930	429.2

TABLE II
COMPARISON WITH CHORTLE-crf AND MIS-pga

5-LUT Mapping Result Comparison: FlowMap-r vs. Other Algorithms for Area Minimization						
Circuit	FlowMap-r		Chortle-crf		Mis-pga	
	LUTs	Depth	LUTs	Depth	LUTs	Depth
5xp1	23	3	27	4	26	4
9sym	61	5	65	8	65	8
9symml	58	5	62	7	65	7
C499	129	6	141	8	123	7
C880	170	11	172	13	172	11
alu2	125	12	128	13	127	15
alu4	223	13	231	17	234	16
apex6	218	7	235	6	221	6
apex7	78	5	78	6	72	5
count	58	4	58	5	59	5
des	930	9	981	10	-	-
duke2	148	6	152	7	161	7
misex1	15	2	18	4	16	3
rd84	37	6	41	7	40	6
rot	209	10	214	11	203	11
vg2	38	4	39	5	37	5
z4ml	13	3	13	4	10	3
total	2533	111	2655	135	-	-

on average resulted in 6% fewer LUT's in the final mapping solution for one-level depth relaxation showed in Table I.

Finally, we have tested the effectiveness of the two post-processing steps, namely MP-Pack and Flow-Pack, that are performed after DF-Map to further minimize the area of the mapping solution by necessary node duplications. Without such post-processing steps, the total number of LUT's used for the designs in Table III is 3087. It is reduced to 2931 after the post-processing. This yields an overall area reduction of 5%. Considering the fact that the percentage of multifanout nodes

TABLE III
COMPARISON WITH FLOWMAP, MIS-pga (delay), AND CHORTLE-d

5-LUT Mapping Result Comparison: FlowMap-r vs. Other Algorithms for Depth Minimization								
Circuit	FlowMap-r		FlowMap		Mis-pga(delay)		Chortle-d	
	LUTs	Dpt	LUTs	Dpt	LUTs	Dpt	LUTs	Dpt
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	134	5	154	5	199	8	382	6
C880	206	8	232	8	259	9	329	8
alu2	149	8	162	8	122	6	227	9
alu4	253	10	268	10	155	11	500	10
apex6	232	4	257	4	274	5	308	4
apex7	83	4	89	4	95	4	108	4
count	76	3	76	3	81	4	91	4
des	1109	5	1308	5	1397	11	2086	6
duke2	188	4	187	4	164	6	241	4
misex1	15	2	15	2	17	2	19	2
rd84	47	4	43	4	13	3	61	4
rot	246	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	2931	83	3261	83	3182	90	4906	87

is much larger than this, it further justified the assumption that an area-optimal mapping solution should not have excessive number of node duplications.

VII. CONCLUSION AND EXTENSION

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 K-bounded general Boolean network, which is used for area 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. Such a generalization was shown *in* [6].

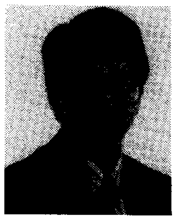
During depth relaxation, we use only structural information to decompose the LUT's. It is also possible to use Boolean optimization techniques to re-synthesize the LUT network locally to explore more possibilities, at the expense of longer computation time. Such a mapping-directed resynthesis technique was used in [5] for further depth optimization, and achieved very promising results.

Acknowledgment

We thank Dr. K. C. Chen and Dr. B. Preas for their helpful discussions.

References

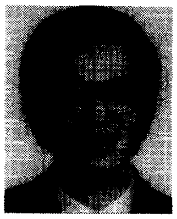
[1] N. Bhat and D. Hill, "Routable technology mapping for FPGA's," *1st Int. ACM/SIGDA Wkshp. on Field Programm. Gate Arrays*, pp. 143–148, Feb. 1992.
[2] R. K. Brayton, R. Rudell, and A. L. Sangiovanni-Vincentelli, "MIS: A multiple-level logic optimization," *IEEE Trans. CAD*, pp. 1062–1081, Nov. 1987.
[3] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA technology mapping for delay optimization," *IEEE Design and Test of Comput.*, pp. 7–20, Sept. 1992.
[4] J. Cong and Y. Ding, "An optimal technology mapping fo delay optimization in lookup-table based FPGA designs," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 48–53, Nov. 1992.
[5] J. Cong and Y. Ding, "Beyond the combinational limit in depth minimization for LUT-based FPGA designs," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 110–114, 1993.
[6] J. Cong, Y. Ding, T. Gao, and K. Chen, "An optimal performance-driven technology mapping algorithm for LUT based FPGA's under arbitrary net-delay models," *Proc. 1993 Int. Conf. on CAD and Computer Graphics*, pp. 599–603, Aug. 1993.
[7] T. Cormen, C. Leiserson, and R. Rivest, *Algorithms*. Cambridge, MA: MIT Press, 1990.
[8] A. Farrahi and M. Sarrafzadeh, "On the lookup-table minimization problem for FPGA technology mapping," in *Tech. Rep. 93-AC-102*, Dept. of EECS, Northwestern Univ. (July 1993).
[9] R. J. Francis, J. Rose, and Z. Vranesic, "Ghortle-crf: Fast technology mapping for lookup table-based FPGA's," *Proc. 28th ACM/IEEE Design Automat. Conf.*, pp. 613–619, June, 1991.
[10] ——, "Technology mapping for lookup table-based FPGA's for performance," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 568–571, Nov. 1991.
[11] D. Hill, "A CAD system for the design of field programmable gate arrays," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 187–192, June, 1991.
[12] K. Karplus, "Xmap: A technology mapper for table-lookup field-programmable gate arrays," *Proc. 28th ACM/IEEE Design Automat. Conf.*, pp. 240–243, June, 1991.
[13] R. Murgai, Y. Nishizaki, N. Shenoy, 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] R. Murgai, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli, "Performance directed synthesis for table look up programmable gate arrays," *Proc. IEEE Int. Conf. on Compuer-Aided Design*, pp. 572–575, Nov. 1991.
[15] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved logic synthesis algorithms for table look up architectures," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 564–567, Nov. 1991.
[16] P. Sawkar and D. Thomas, "Technology mapping for table-look-up based field programmable gate arrays," *1st Int. ACM/SIGDA Wkshp. on Field Programm. Gate Arrays*, pp. 83–88, Feb. 1992.
[17] M. Schlag, J. Kong, and P. K. Chan, "Routability-driven technology mapping for lookup table-based FPGA's," *Proc. 1992 IEEE Int. Conf. on Computer Design*, pp. 86–90, Oct. 1992.
[18] N.-S. Woo, "A heuristic method for FPGA technology mapping based on the edge visibility," *Proc. 28th ACM/IEEE Design Automat. Conf.*, pp. 248–251, June, 1991.
[19] Xilinx, *The Programmable Gate Array Data Book*. Xilinx: San Jose, 1992.



Jason Cong (M'90) received the B.S. degree in computer science from the Peking University in 1985 and the M.S. and Ph.D. degrees, also in computer science, from the University of Illinois at Urbana-Champaign in 1987 and 1990, respectively.

Currently, he is an Assistant Professor with the Computer Science Department, University of California, Los Angeles. From 1986 to 1990, he was a research assistant with the Computer Science Department of the University of Illinois. He worked at the Xerox Palo Alto Research Center in the summer of 1987 and at the National Semiconductor Corporation in the summer of 1988. His research interests include computer-aided design of VLSI circuits, fault-tolerant design of VLSI systems, and design and analysis of efficient combinatorial and geometric algorithms.

Dr. Cong has served on the program committees of a number of VLSI CAD conferences and chaired the 4th ACM/SIGDA Physical Design Workshop. He received the Best Graduate Award from the Peking University in 1985. He was awarded a DEC Computer Science Fellowship in 1988, received the Ross J. Martin Award for Excellence in Research from the University of Illinois at Urbana-Champaign in 1989, the NSF Research Initiation Award in 1991, and the NSF Young Investigator Award in 1993. He also received the Northrop Corporation Outstanding Junior Faculty Research Award from the UCLA School of Engineering and Applied Sciences in 1993.



Yuzheng Ding was born in Qingdao, China. He received the B.S. degree in computer science from Peking University, and the M.S. degree in computer science from Tsinghua University, both in Beijing, China.

Currently he is a Research Assistant with the Department of Computer Science, University of California, Los Angeles, where he is pursuing the Ph.D. degree. His research interests include computer-aided design of VLSI circuits, design and analysis of data structures and algorithms, and database systems.

Mr. Ding is a member of ACM.