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

Abstract

In this paper we study the area and depth trade-off in 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 and advantageous to subsequent re-mapping for area minimization. We then re-map the resulting network to obtain an area-minimized mapping solution. 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. For the area minimization step, we have developed an optimal algorithm for computing an area-minimum mapping solution without node duplication. Experimental results 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. The lookup table (LUT) based FPGAs are produced by several FPGA manufacturers [16, 8], in which the basic programmable logic block is a K-input lookup table that 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 designs can be roughly divided into three categories according to their optimization objectives: the area minimization algorithms [6, 10, 12, 9, 15, 13], the depth minimization algorithms [7, 11, 13, 2, 3], and the algorithms that maximize routability [14, 1]. Most of these mapping algorithms are based on heuristic techniques, except FlowMap [3] which guarantees to produce depth-optimal mapping solutions. It remains open if the area-optimal mapping problem for LUT-based FPGAs can be solved efficiently.

The common limitation of the existing algorithms is 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. Figure 1 compares the 5-LUT mapping results by some existing algorithms on an MCNC benchmark circuit named *rot*. The depths and the numbers of LUTs of the solutions by different algorithms vary significantly. In general, the area-minimized solutions have much larger depth, while the depth-minimized solutions use much more LUTs. However, in practice the best design may not come from either 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 the 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 [3]), 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.

Due to page limitation, detailed algorithm descriptions and the proofs of the theorems are not included in this paper. They can be found in [4].

2. Problem Formulation

A general Boolean network is represented as a DAG where a 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 number of edges on the longest path from any PI node to v. The *depth* of a network is the largest node level in the

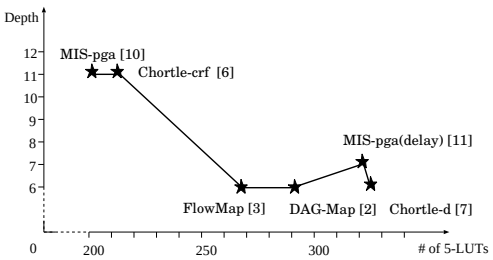


Figure 1 Mapping solutions of various algorithms for *rot* (K=5).

network. A Boolean network is *K-bounded* if $|input(v)| \le K$ for each node v. In the rest of this paper, we only consider 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 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 $|input(C_v)| \le K$.

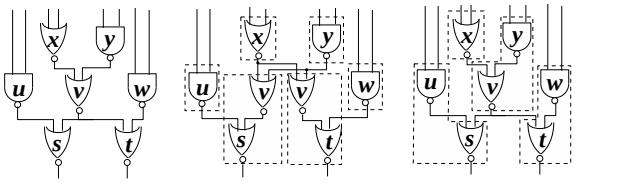
If a K-LUT LUT_v implements (*covers*) a K-feasible FFC of v, we say that LUT_v *implements* node v and that v is the *root* of LUT_v. If the K-feasible cone C_v is not fanout free, we have to duplicate the non-root nodes in C_v that have fanouts outside of C_v in order to cover C_v by a K-LUT. Given 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 DAG 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 satisfied 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 discuss the effect of *depth relaxation* and *node duplication* which are important in area/depth trade-off, and give an overview of our algorithm.

FlowMap [3] produces depth-optimal LUT-based FPGA mapping solutions for general Boolean networks. It worths noticing that in a FlowMap mapping solution, every node (LUT) has the minimum possible depth. Insisting minimum depth for every node, including the non-critical ones, may lead to inefficient use of LUTs. Figure 3(b) shows the mapping solution by FlowMap for the network in Figure 3(a). Another solution is shown in Figure 3(d), which has the same depth as (b) but uses one fewer LUT. Note that LUT_v in (b) has the minimum depth 3. 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



(a) original network (b) mapping with node v duplicated (6 LUTs, 2 levels) (c) mapping without node duplication (4 LUTs, 3 levels)

Figure 2 Technology mapping for LUT-based FPGA (K=3).

[1] We use the DMIG algorithm [2] to transforms a network which is not K-bounded into a K-bounded one with minimum depth.

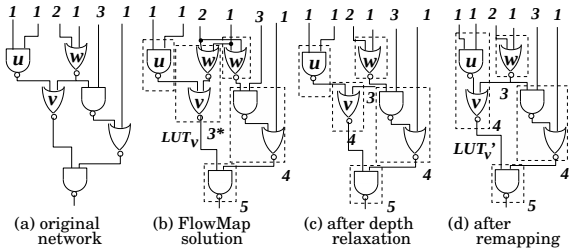


Figure 3 Depth relaxation for area reduction (K=3, numbers mean levels)

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. More discussions about depth relaxation will be given in section 4.

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 node v in Figure 2(a)(b)). In general, node duplication is very important to depth optimization. Without node duplication, we may have to 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 very often used to guarantee the optimal depth. 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 of the m LUTs. Therefore, excessive node duplication will very likely result in large number of LUTs. Based on this observation, we have developed the DF-map algorithm, which performs area-optimal mapping without node duplication. Details will be discussed in section 5.

Our algorithm, named FlowMap-r, 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. During 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. Then, we carry out two post-processing procedures which allow necessary node duplications for further area minimization. The two procedures are MP-Pack, a matching-based multi-fanout predecessor packing procedure from the DAG-Map package [2], and Flow-Pack, a flow-based area minimization procedure from the FlowMap package [3].

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.

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

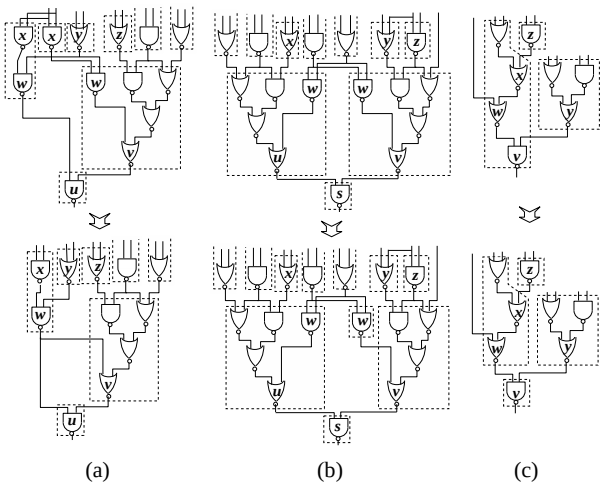


Figure 4 Three types of depth relaxation operations (assume K=5 and LUT_v has non-zero slack).

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/or LUT_w can be packed with existing LUTs during subsequent re-mapping (as shown in Figure 3(d)). Since the depth relaxation operations may lead to different results when applied on different LUTs, we want to apply them to the most 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, the 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. 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 to exclude its copy of node w, no new LUT needs to be generated.

In Figure 4(c), LUT_v contains node w which has a single fanout. However, decomposing 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 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.

It is easy to see that 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 an algorithm for area-optimal mapping without node duplication (*duplication-free mapping*, or *DF-mapping*) for 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 DF-mapping uses 2 LUTs, while tree-based mapping uses 6 LUTs. Our algorithm is based on an important concept called the *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 $MFFC_w \subseteq MFFC_v$. □

Lemma 2 Two MFFCs are either disjoint or one must contain the other. □

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

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

First, we point out 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 $MFFC_v$. □

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

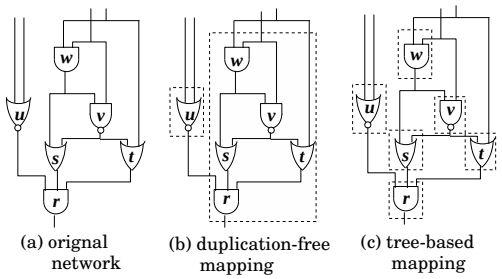


Figure 5 Duplication-free mapping vs. tree-based mapping (K=3).

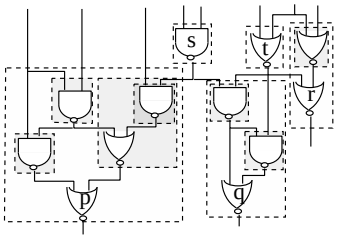


Figure 6 Maximum fanout free cones.

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.

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 the least cost. Cost computation of each cut involves recursively solving a set of DF-mappings for MFFCs of smaller sizes. Clearly, the same discussion can be applied to depth-optimal DF-mapping by simply altering the cost function.

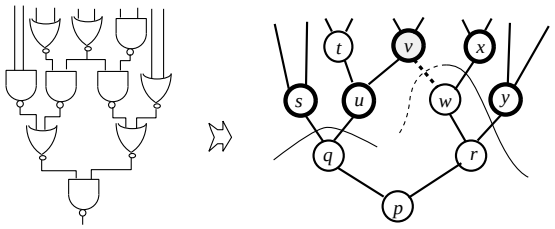
It is not difficult to see that there are only polynomial number of K-feasible cuts, since the total number of possible combinations of K or fewer nodes is $O(n^K)$, where n is the number of nodes in the MFFC. In practice, however, examining all these combinations to compute the K-feasible cut with the least cost is too wasteful, since most of them do not form a K-feasible cut. We shall present a more efficient way to generate the K-feasible cuts in $MFFC_v$.

We first consider the case where the MFFC is a tree. 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. Based on this fact, we can generate all K-feasible cuts of a tree recursively. Note that in this case, the number of cuts generated according to this recursion is bounded by a constant that depends only on K and is independent of the size of $MFFC_v$. More specifically, we can prove

Lemma 4 If $MFFC_v$ is a tree, the number of cuts of size K in $MFFC_v$ is no more than the Kth *Catalan number*, i.e. $\frac{1}{K}\binom{2K-2}{K-1}$. □

For K=5 the above bound is 14.

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



(a) (b)

Figure 7 Complication in cut generation.

$v_1, v_2, ..., v_f$ $(f \le K)$, 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. If we represent a cut $(X, \bar{X})$ by $input(\bar{X})$, then, $\{s,u\}$ is a cut in the left subtree, and $\{x,y\}$ is a cut in the right subtree, but their combination does not form a cut in the MFFC, since the edge $<v, w>$ provides a path connecting the nodes outside of the MFFC to the root p. On the other hand, the cut $\{s,u,v,x,y\}$ of the MFFC cannot be generated from the combinations of the cuts in the two subtrees, since $\{s,u,v\}$ 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 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, $\{s,u,v\}$ is a generalized cut of the left subtree.

It can be shown (see [4] for details) that the generalized cuts of tree T can be generated from the combinations of the generalized cuts of its subtrees $T_1, T_2, \ldots, T_f$, and the combinations of the the root with the escape nodes. Both the set of generalized cuts and the set of escape nodes can be computed recursively. Therefore, we can compute all the generalized cuts of size no more than K 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. However, our experimental results showed that in practice, the recursion often quickly reaches the point where the subtree does not contain any escape node. In this case, the more efficient tree cut generation algorithm is applied. For $K = 5$, the number of all possible K-node combinations is $O(n^5)$, while the total number of cuts examined by our algorithm is between n^2 to n^3, where n is the number of nodes in one MFFC.

We can further improve the efficiency of the DF-mapping algorithm by collapsing 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. □

In our experiments, such collapsing reduces the network size by 25% to 50% (when $K = 5$).

Our area-optimal DF-mapping algorithm, the DF-Map, is summarized as follows. We first collapse each K-feasible $MFFC_v$ into node v. 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. Based on the above discussion we have

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

In our FlowMap-r algorithm, unnecessary node duplications by FlowMap are eliminated in the depth relaxation step. The subsequent DF-mapping minimizes area without introducing new node duplication. After applying the DF-Map algorithm, we carry out the two post-processing procedures, the MP-Pack [2] and the Flow-Pack [3], which explore necessary node duplications for further area reduction.

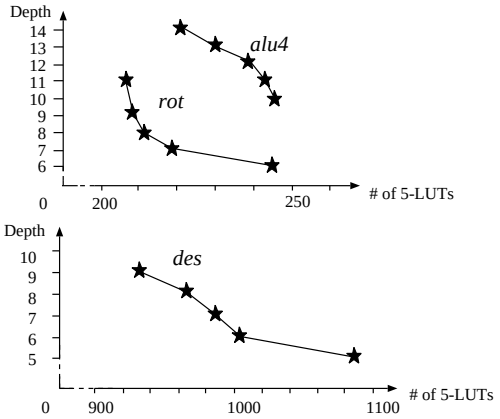


Figure 8 Area/depth trade-off in FlowMap-r (K=5).

FlowMap-r Mapping Results for 5-LUT FPGAs						
Circuit	**Optimal Depth**	**No. of 5-LUTs For Different Depths**				
	d_{opt}	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.

6. Experimental Results

We have implemented the FlowMap-r algorithm on SUN Sparc workstations and tested it on the benchmark circuits used in [7, 2, 3]. Table 1 shows some of the mapping solution sets computed by FlowMap-r. In general, larger networks have more room for area and depth trade-off, as shown in Figure 8. The area reduction is usually more significant at the first a few steps of depth bound increase.

We also compared 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 [7, 3, 11]. 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, FlowMap-r shows better performance than the compared algorithms.[2] Moreover, FlowMap-r is solely based on combinatorial optimization techniques, thus it is fast. Furthermore, FlowMap-r produces a set of mapping solutions, each of them satisfies an explicitly assigned depth bound. Therefore, it gives the designer more choices.

The experiments also show that the reduction on the number of LUTs by the post-processing steps that performs necessary node duplications is 5% to 10%. Since the

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.

[2] The improved version of MIS-pga program, MIS-pga(new) [12], outperforms FlowMap-r in terms of area. The depths of their solutions are not reported in [12].

5-LUT Mapping Result Comparison: FlowMap-r vs. Depth Minimization Algorithms								
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	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	155	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.

percentage of multi-fanout nodes is much larger, it further justifies the assumption that an area-optimal mapping solution should not have large number of node duplications.

7. Conclusion and Future Work

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/depth trade-off. We have developed an efficient method to compute an optimal duplication-free mapping solution for a general network, and used it for area minimization. The concept of the *maximum fanout free cone* plays an important role in our duplication-free mapping algorithm, and it may finds applications to other logic synthesis problems as well. The solution sets generated by our algorithm outperform the solutions by many existing algorithms in terms of area and depth. Although the algorithm is presented under *unit delay* model, it can be generalized to the case where an arbitrary delay is assigned to a net [5].

During depth relaxation, we only use 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-based 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. 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

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

[2] Chen, K. C., J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, ‘‘DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization,’’ *IEEE Design and Test of Computers*, Sep. 1992.

[3] 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.

[4] Cong, J. and Y. Ding, ‘‘On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping,’’ *UCLA Computer Science Department Tech. Report CSD-920053*, October 1992.

[5] Cong, J., Y. Ding, T. Gao, and K. Chen, ‘‘An Optimal Performance-Driven Technology Mapping Algorithm for LUT based FPGAs under Arbitrary Net-Delay Models,’’ *Proc. 1993 Int’l Conf. on Computer Graphics and CAD*, August 1991.

[6] 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.

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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