DAOmap: A Depth-optimal Area Optimization Mapping Algorithm for FPGA Designs

Deming Chen, Jason Cong
Computer Science Department
University of California, Los Angeles
{demingc, cong}@cs.ucla.edu

ABSTRACT
In this paper we study the technology mapping problem for FPGA architectures to minimize chip area, or the total number of lookup tables (LUTs) of the mapped design, under the chip performance constraint. This is a well-studied topic and a very difficult task (NP-hard). The contributions of this paper are as follows: (i) we consider the potential node duplications during the cut enumeration/generation procedure so the mapping costs encoded in the cuts drive the area-optimization objective more effectively; (ii) after the timing constraint is determined, we will relax the non-critical paths by searching the solution space considering both local and global optimality information to minimize mapping area; (iii) an iterative cut selection procedure is carried out that further explores and perturbs the solution space to improve solution quality. We guarantee optimal mapping depth under the unit delay model. Experimental results show that our mapping algorithm, named DAOmap, produces significant quality and run-time improvements. Compared to the state-of-the-art depth-optimal, area minimization mapping algorithm CutMap [21], DAOmap is 16.02% better on area and runs 24.2X faster on average when both algorithms are mapping to FPGAs using LUTs of five inputs. LUTs of other inputs are also used for comparisons.

1. INTRODUCTION

The Field Programmable Gate Array (FPGA) has become increasingly popular throughout the past decade. The cost pressures, changing requirements, and short design windows favor more and more programmable chip solutions. Generally, an FPGA chip consists of programmable logic blocks, programmable interconnections, and programmable I/O pads. The LUT-based FPGA architecture dominates the existing programmable chip industry, in which the basic programmable logic element is a K-input lookup table. A K-input LUT (K-LUT) can implement any Boolean functions of up to K variables. FPGA technology mapping converts a given Boolean circuit into a functionally equivalent network comprised only of LUTs. This phase is followed by a placement and routing phase to realize an implementation of the mapped network. As the routing resources in FPGAs are prefabricated and relatively limited compared to ASIC technologies, the technology mapping process has a significant impact on the area, performance and power of the implemented circuit.

Previous technology mapping algorithms for LUT-based FPGA designs can be roughly divided into four categories according to their optimization objectives.

- **Area Minimization**. These algorithms include Chortle-crf [1], MIS-pga [2], XMap [3], VisMap [4], TechMap [5], and Praetor [6]. It is an *NP*-hard problem to solve for general circuits as DAG's (directed acyclic graphs) [7].
- **Delay Minimization**. These algorithms include Chortle-d [8], MIS-pga-delay [9], TechMap-L [5], DAG-map [10], FlowMap [11], EdgeMap [12], and SeqMapII [13]. FlowMap is the first algorithm to guarantee a depth-optimal mapping solution in polynomial time under the *unit delay* model. EdgeMap presents a non-trivial generalization of FlowMap and achieves optimal delay using the general delay model, where each interconnection edge has a weight representing the delay of the interconnection.
- **Power Minimization**. These algorithms include mapping algorithms in [14] and [15], PowerMap [16], PowerMinMap [17], Emap [18], and DVmap [19]. Min-power mapping is also a *NP*-hard problem to solve [14].
- **Delay and Area Minimization**. The algorithms in this category are particularly interesting because they minimize both delay and area of the FPGA design. The major works include FlowMap-r [20] and CutMap [21], where no logic resynthesis/restructuring is performed. Other works, such as BoolMap [22], FlowSYN [23] and the work in [24], consider logic resynthesis during mapping.

Techniques used in the above-mentioned mapping algorithms include bin packing, dynamic programming, greedy algorithm, binate covering, network flow algorithm, BDD-based logic optimization, and cut-enumeration algorithm, etc. For FlowMap-r and CutMap, both algorithms target minimizing area while maintaining optimal mapping depth. FlowMap-r started with the depth-optimal mapping solution produced by FlowMap and then applied depth-relaxation techniques such as remapping and node packing for the non-critical paths. It reported better area than FlowMap, and better area and mapping depth than MIS-pga-delay and Chortle-d. CutMap combined depth and area minimization during the mapping process by computing min-cost min-height K-feasible *cuts* for critical nodes and computing min-cost K-feasible cuts for non-critical nodes using the network flow method. It reported better area than FlowMap-r with the same mapping depth and similar run-time. CutMap is widely used in the academic community for various FPGA evaluation and design flows, and has been considered the state-of-the-art FPGA technology mapping algorithm for depth-optimal mapping solutions.

In this paper we present a new mapping algorithm, DAOmap, for Depth-optimal Area Optimization of FPGA designs. We adopt a cut-enumeration-based method that consists of cut generation and cut selection. Cut generation traverses the network from primary inputs (PIs) to primary outputs (POs), and combines subcuts on the fanin nodes of the target node to generate all the cuts on the target node (each cut represents one possible LUT implementation rooted on the target node). After all the cuts are generated, we traverse the network from POs to PIs and select cuts to produce the LUT mapping result. The difficulty lies in the way we select a subset of all the cuts to cover the whole circuit. A binate-covering algorithm will give an optimal solution but with exponential complexity. To design effective heuristics to select cuts, we have to consider the global view of the area optimization because early selections of cuts will have a strong impact on the future selections of cuts during the traversal from POs to PIs. Through the mapping procedure, node duplication cannot be avoided if the optimal mapping depth is to be guaranteed. Actually, how to handle the different duplication scenarios makes the min-area FPGA mapping problem *NP*-hard. In this work we offer three novel approaches to effectively model and control node duplications and reduce area through the entire mapping process. First, we consider the potential duplications during the cut generation procedure so the mapping solutions encoded in the cuts consider duplication costs. This will help the cut selection procedure to make the right decisions to cover the circuit with less node duplications from a global optimization point of view. Second, after the timing constraint is determined (the longest optimal mapping delay of the network), we will relax the non-critical paths by searching the solution space considering both local and global optimality information to minimize the mapping area. Third, we carry out an iterative cut selection procedure that further explores and perturbs the solution space to improve solution quality. Experimental results show that our algorithm produces significant quality and run-time improvements over CutMap across a series of MCNC benchmarks and a set of industrial benchmarks.

The rest of this paper is organized as follows. In Section 2 we provide some basic definitions and formulate the depth-optimal area optimization problem for FPGA. Section 3 gives a review of the cut-enumeration-based mapping algorithm. Section 4 shows the detailed description of our enhancement of cut enumeration in terms of cost estimation and other novel techniques to minimize area. Section 5 presents experimental results, and Section 6 concludes this paper.

2. DEFINITIONS AND PROBLEM FORMULATION

A Boolean network can be represented by a DAG where each node represents a logic gate, and a directed edge *(i, j)* exists if the output of gate *i* is an input of gate *j*. A PI node has no incoming edges and a PO node has no outgoing edges. We treat the flip-flop outputs as special PIs and the flip-flop inputs as special POs, and make no distinction in terms of notation. We use *input(v)* to denote the set of nodes which are *fanins* of gate *v*. Given a Boolean network *N*, we use O_v to denote a *cone* rooted on node *v* in *N*. O_v is a sub-network of *N* consisting of *v* and some of its predecessors, such that for any node $w \in O_v$, there is a path from *w* to *v* that lies entirely in O_v. The maximum cone of *v*, consisting of all the PI predecessors of *v*, is called a *fanin cone* of *v*, denoted as F_v. We use $input(O_v)$ to denote the set of distinct nodes outside O_v which supply inputs to the gates in O_v. A *cut* is a partitioning *(X, X')* of a cone O_v such that *X'* is a cone of *v*. The *cut-set* of the cut, denoted *V(X, X')*, consists of the inputs of cone *X'*, or *input(X')*. A cut is *K-feasible* if *X'* is a *K*-feasible cone. In other words, the cardinality of the cut-set is $\leq K$. We also call the cardinality of the cut-set the *cutsize* of the cut. The *level* of a node *v* is the length of the longest path from any PI node to *v*. The level of a PI node is zero. The *depth* of a network is the largest node level in the network. A Boolean network is *l-bounded* if $|input(v)| \leq l$ for each node *v*.

Because the exact layout information is not available during the technology mapping stage, we model each interconnection edge in the Boolean network as having a constant delay. Therefore, we approximate the largest delay of the mapped circuit with a *unit delay* model, where each LUT on the critical path (the path with longest delay after mapping) contributes one unit delay. The largest optimal delay of the mapped circuit is also called the optimal *mapping depth* of the circuit.

The mapping problem for depth-optimal area optimization of FPGA is to cover a given *l*-bounded Boolean network with *K*-feasible cones, or equivalently, *K*-LUTs in such a way that the total LUT count after mapping is minimized while the optimal mapping depth is guaranteed under the unit delay model. Our initial networks are all 2-bounded and *K* can be 4, 5 or 6 in this study. Therefore, our final mapping solution is a DAG in which each node is a *K*-LUT and the edge (O_u, O_v) exists if *u* is in $input(O_v)$. Our algorithm will work for any reasonable *K* values.

3. REVIEW OF CUT ENUMERATION

3.1 Cut Enumeration with Delay and Area Propagation

Cut enumeration is an effective method for finding all the possible ways of the *K*-feasible cones rooted on a node [6, 19, 20]. A cut rooted on node *v* can be represented using a product term (or a *p-term*) of the variables associated with the nodes in the cut-set $V(X_v, X_v')$. A set of cuts can be represented by a sum-of-product expression using the corresponding p-terms. Cut enumeration is guided by the following theorem [6]:

$$f(K, v) = \otimes^{K}_{u \in input(v)} [u + f(K, u)] \qquad (1)$$

where *f(K, v)* represents all the *K*-feasible cuts rooted at node *v*, operator + is Boolean *OR*, and $\otimes^K$ is Boolean *AND* on its operands, but filtering out all the resulting *p-terms* with more than *K* variables. We will use a simple example to illustrate the cut-enumeration process. We use $\{s_1, \ldots, s_n\}$ to represent a cut with cut-set $s_1, \ldots, s_n$, where s_i is either an internal signal or a PI. In Figure 1, all the cuts[1] rooted on node *s* can be generated by combining the cuts rooted on its *fanin nodes q* and *r* (refer to Formula (1)). We call the cuts on the fanin nodes *subcuts*. Combining C_1 with C_2 will form a new cut $C_s = \{m, n, o, p\}$ rooted on *s* (shaded area). The cut-enumeration process will combine each subcut (or the fanin node itself) on one of the fanin nodes with each counterpart from the other fanin node to form new cuts for the root node. If the input of the new cut exceeds *K*, the cut is discarded. During this enumeration process, the arrival time and mapping cost/area for each cut can be calculated (we use the words *cost* and *area* interchangeably). Subsequently, the arrival time and area for the root node can be obtained as well.

The arrival time of PI nodes is 0. The arrival time will propagate through the cuts from PIs to POs, where each cut (LUT) on the paths represents one unit delay. To get the minimum arrival time for a node *v*, we have

[1] We will use the term *cut*, and its symbol *C*, to represent the *K*-feasible *cone* that is a result of this cut for simplicity purpose.

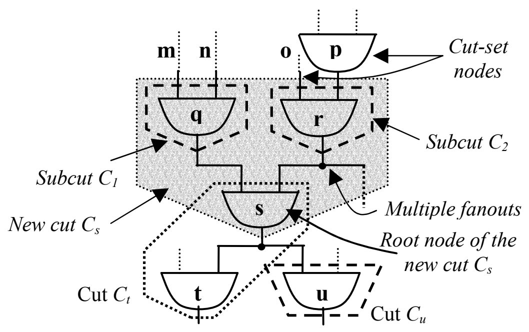

Figure 1: Cut generation

$$Arr_v = \underset{\forall C \text{ on } v}{\text{MIN}} \ [\underset{i \in input(C)}{\text{MAX}} (Arr_i) + 1] \tag{2}$$

where C represents every cut generated for v through cut enumeration. Here, the arrival time of C is $MAX(Arr_i) + 1$, where Arr_i is the minimum arrival time on input signal i of C. We call the cut C that provides Arr_v as MC_v. Notice that there can be more than one MC_v for node v, and all the MC_v's form a set X_v. Thus, the minimum arrival time for each node in the network is propagated from the PIs through cuts and iteratively calculated until all the POs are reached by a topological order. The longest minimum arrival time of the POs is the minimum arrival time of the circuit, i.e., the optimal mapping depth of the circuit.

Similarly, the area can be propagated along the process of cut enumeration. The area for a cut C is calculated as follows:

$$A_C = \sum_{i = input(C)} [A_i / f(i)] + U_C \tag{3}$$

where U_C is the area contributed by cut C itself (to be covered later), A_i is the estimated area of a cone rooted on signal i, and $f(i)$ is the fanout number of signal i. Therefore, the area on i is shared and distributed into other fanout nodes of i. Once the outputs of i reconverge, the total area on i will be summed up. This process is trying to estimate the area more accurately, considering the effects of gate fanouts [6]. The cut that gives the best A_C on node v is the *effective cost* of v. The area propagation in [6] was based on sub-networks, such as MFFCs (*maximum fanout free cones*[2]), where nodes outside of the target MFFC did not contribute area to A_C, and $U_C = 1$ always. It was area-oriented and did not guarantee optimal mapping depth. This area estimation model would represent the actual mapping area for duplication-free mapping based on MFFCs and would be a lower bound of the minimum mapped area if node duplication was allowed [6]. We will show that this conclusion on the lower bound no longer holds with the optimal mapping depth constraint in Section 4.

3.2 Complexity Analysis

The number of cuts on a node for the worst case is $O(n^K)$, where n is the total number of nodes in the network. However, when LUT input K is small, the number of cuts generated for each node v is a small constant because all the cuts are usually generated within a small cone O_v contained in F_v. Figure 2 shows this situation,

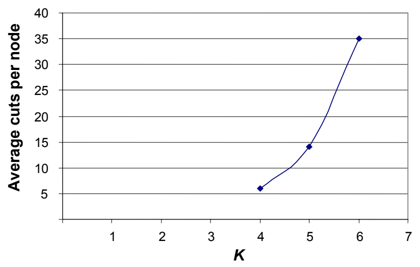

Figure 2: Average number of cuts on each node along K

where the vertical coordinate shows the number of cuts generated for each node averaging over the 20 largest MCNC benchmarks along the value of K. This implies that the complexity of cut-enumeration algorithm is practically linear to n for small K. Nonetheless, the number of cuts can be large when $K \geq 7$. For large K, we can apply *cut-pruning* techniques to control the number of cuts to be generated. Since this is not the focus of this work, interested readers are referred to [6].

4. ALGORITHM DESCRIPTION

Based on the cut-enumeration framework, we first present our enhancements in terms of area propagation (Section 4.1), cost function (Section 4.2), and global duplication cost adjustment (Section 4.3). Then we present a series of novel techniques to guide the cut selection procedure (Section 4.4).

4.1 Area Propagation under Timing Constraints

To guarantee optimal mapping depth, we need to propagate the estimated area together with the propagation process of the minimum arrival time. Thus, A_i of Formula (3) in our case will be the best propagated area in the fanin cone F_i. Then, after we have calculated the area for each cut rooted on node v, the best propagated area in the fanin cone F_v is as follows:

$$A_v = \underset{C \in X_v}{\text{MIN}} (A_C) \tag{4}$$

A_v represents the best achievable mapping area up to node v under the constraint that it also generates the optimal mapping delay up to the point of v. Through Formula (3) and (4), the areas of the cuts and nodes are iteratively calculated until the enumeration process reaches all the POs. Later on, during the cut selection stage when we know that v is not on a critical path, a cut $C \notin X_v$ may be picked as long as it will not violate the timing constraint and will produce a better area in the same time. Thus, A_v represents an area estimation guarded by a timing constraint that usually is overstressed unless v is on a critical path. Nonetheless, we have to stick to A_v during cut enumeration because we do not know whether v is a critical node or not until the entire cut-enumeration process is completed. Therefore, $\Sigma_{v \text{ is a PO}} (A_v)$ is no longer a lower bound on the minimum area of all mapping solutions under the optimal mapping depth constraint.

4.2 Cost Function for a Cut

Although each cut represents one LUT, using a fixed unit area for a cut will not accurately reflect the property of the cut. Figure 3 illustrates this situation. Although both cut C_1 and cut C_2 have a cutsize of 2, C_2 is obviously a better cut because it covers many more nodes than C_1, thus implementing more logic. We also

[2] For any node $u \neq$ root node v, if u is contained in the cone, all of the outputs of u are also in the cone. On the other hand, if all of the outputs of u are in the cone, then u is also in the cone [20].

observe that C_2 covers two sets of reconvergent paths completely, which is the reason it can cover six nodes with a small cutsize. Therefore, the number of all the reconvergent paths covered by a cut is also in the cost function.[3] The third factor we consider is the fanout number of the root node. The larger this fanout number, the larger the possibility that picking this cut will reduce potential duplications. For example, node 5 has four fanouts. Picking cut C_2 or C_3 will cause node 5 to be duplicated multiple times when each of its other fanout nodes (on dashed lines) is mapped. However, picking a cut rooted on node 5 itself will reduce these potential duplications. Finally, cuts of different cutsizes have different areas. The number of edges in a mapped network is on the order of O(*Kn*), where *n* is the number of LUTs in the network. If the number of edges is minimized, it naturally minimizes the number of LUTs as well. This indicates that always preferring LUTs with large input numbers will hurt the area from the overall picture. We use the following formula to calculate the area of a cut *C*:

$$U_C = \frac{\alpha \cdot I_C}{N_C + \beta \cdot (f(v) + R_C)} \qquad (5)$$

where I_C is the cutsize of *C*, N_C is the number of nodes covered by *C*, $f(v)$ is fanout number of the root node *v*, R_C is the number of reconvergent paths completely covered by *C*. α and β are positive constants (we set $\alpha = 0.8$ and $\beta = 0.4$). The smaller the value of U_C, the better the cost of *C* is.

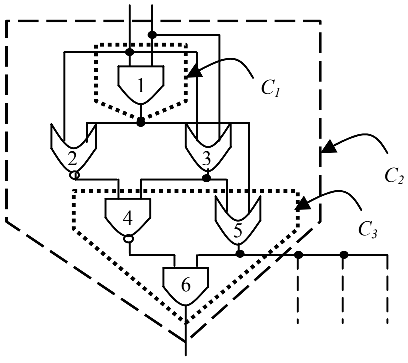

Figure 3: Illustration of various cuts

4.3 Global Duplication Cost Adjustment

It is intuitive that the quality of the mapping result is directly related to the accuracy of area estimation. Although the area model presented in the previous section no longer represents a lower bound of the mapped area, it has obvious advantages. Using the example shown in Figure 1, if C_s is used to implement an LUT in the final mapping and there is no duplication involved, the area rooted on node *s*, A_s, should be equally shared by fanouts *t* and *u*. Otherwise, A_s will be falsely double-counted when it is propagated to both *t* and *u* later. However, this estimation has its downside. Suppose the final mapping result uses C_t and C_u (Figure 1), then the estimation is no longer accurate because C_u treats node *s* as not duplicated[4] but *s* is actually duplicated in C_t. Thus, the area model can under-estimate the actual mapping area.

[3] Starting from the root node, do a breadth-first search toward the cut-set nodes to find the reconvergent nodes, such as node 3 in the example.

[4] The area A_s is divided by 2 and propagated to C_u.

To address this issue during cut enumeration, we adjust the estimated area according to the potential node duplication scenarios. These can be captured by checking the subcuts that are with or without multiple fanouts. In Figure 1, when subcuts C_1 and C_2 are combined to form C_s, we observe that there will not be any node duplications for node *q* because it is a single-fanout node. However, there will be a duplicated node for node *r* since it has another fanout (dashed line) that goes out of cut C_s. We change Formula (3) to the following:

$$A_C = \sum_{i = input(C)} [A_i / f(i)] + U_C + P_{f1} + P_{f2} \qquad (6)$$

where f_1 and f_2 are the two fanin nodes of the root node *v*, and *C* is formed by the two subcuts C_{f1} and C_{f2} rooted on f_1 and f_2 respectively. P_{f1} and P_{f2} are the duplication costs of the subcuts, which are defined respectively as follows:

$$P_f = \begin{cases} N_{Cf} / I_C & \text{if } f(i) > 1 \\ 0 & \text{otherwise} \end{cases} \qquad (7)$$

where *C* is defined as above, C_f is the subcut on either f_1 or f_2, and $f(i)$ is the fanout number of corresponding f_1 or f_2. N_{Cf} is the number of nodes C_f contains, and I_C is the cutsize of *C*. The intuition is that the larger N_{Cf} is, the larger the possibility that more nodes in C_f will be duplicated, thus the larger duplication cost it produces. I_C is treated as a normalizing factor because the larger I_C is, the more likely it is that *C* will contain more nodes. This, in a way, alleviates the duplication cost experienced in the local area of the cut, i.e., in C_f. Notice, once A_C is adjusted it stays that way, and the related node cost will also consequently be adjusted. Meanwhile, the new cost will propagate to the cuts and nodes on the fanouts. In Figure 1, once the costs of C_s and A_s are adjusted, they will start to influence the costs of C_u and A_u, etc. Thus, this cost adjustment has a global impact for the area propagation process and makes the area estimation more closely related to the actual mapping implementation in a global point of view. This will improve the quality of the final mapping, as shown in the section of experimental results.

4.4 Cut Selection

After cut enumeration, we obtain the optimal mapping depth of the network. This mapping depth will be set as the required time for the circuit. The critical path(s) is the path that leads to such a mapping depth, and the nodes on non-critical paths will have the luxury of selecting different cuts that offer smaller costs with a relaxed delay value as long as the required time of the circuit is maintained. This is a standard technique for area minimization under timing constraints. As mentioned in the Introduction section, we will carry out a topological order traversal starting from POs, then the inputs of the generated LUTs are iteratively mapped. The procedure continues until all the PIs are reached.

As mentioned before, the difficulty of mapping lies in the method of selecting cuts to cover the whole circuit to minimize the total area. We cannot greedily pick the cuts with the smallest costs calculated through the cut-enumeration process, because that will forfeit some important optimization factors in terms of reducing node duplications locally. Therefore, we design several novel heuristics that will search the solution space, guided by both local and global optimization information to increase area savings. We also carry out an iterative cut selection procedure that further explores and perturbs the solution space to improve solution quality. The iterative cut selection procedure will be introduced first because it is in the outer loop of the optimization cycle.

4.4.1 Iterative Cut Selection Procedure

We design an iterative cut selection (*ICS*) procedure that produces a final mapping result based on previous cut selection iterations. A previous iteration can be considered as a tentative mapping that provides guidance for the next iteration. This iterative procedure offers extra opportunities to reduce duplications (as shown later). It is intertwined with local cost adjustment techniques (Section 4.4.2). Figure 4 provides a high-level description of the algorithm.

```
algorithm ICS
input: network, output: mapping solution S.
for each iteration do
    L := list of POs; S := $\phi$;
    req_time($v \mid v \in L$) := optimal_mapping_depth;
    while $L \neq \phi$ do
        remove a node v from L;
        $LUT_v$ := pick_cut(v);
        $S := S \cup \{ LUT_v \}$;
        update_profiling_info($LUT_v$);
        update_req_time(input($LUT_v$));
        $L := L \cup \{ input(LUT_v) \}$;
    end-while
end-for
output S;
```

Figure 4: Algorithm of iterative cut selection (ICS)

The early iterations will offer profiling information for the subsequent iteration. The profiling information includes the *LUT roots* (updated through list *L*), which were the selected LUT root nodes (LUT outputs) for the LUT_v's in the mapping solution *S* of the previous iteration. It also includes the list of nodes that were duplicated in the previous iteration. A duplicated node can be found when more than one LUT is covering the node. The profiling information is collected in *update_profiling_info*. Subroutine *pick_cut* will use the profiling data and some other criteria to update the cost of each cut rooted on *v* for the current iteration. *update_req_time* will update the required times of the input nodes of the selected LUT_v through the following formula:

$$Req_i = \underset{i \in input(LUT_v) \text{ for all } v}{\text{MIN}} (Req_v - 1) \qquad (8)$$

where Req_i is the required time of signal *i* that may have multiple fanout nodes. The required times on the fanouts, Req_v's, are already decided following topological order before finalizing the value of Req_i.

4.4.2 Local Cost Adjustment

To map a critical node *v*, only the cut that provides A_v (Formula (4)) is picked to implement the LUT to guarantee the optimal mapping depth. How to select the cuts for the non-critical nodes thus becomes the key to reduce mapping area. To increase the chance of duplication reduction, we will not directly pick the cut with the best cost calculated globally by the cut-enumeration process. Instead, the cost of the examined cut will be adjusted depending on the characteristics of the cut itself (thus the term *local*). After the cost adjustment, it is possible that another cut with an originally unfavorable cost becomes the most favorable to map the current node. We will introduce three techniques below.

Input Sharing. During the cut selection procedure, LUT roots are either in list *L* waiting to be mapped or have been removed from *L* and mapped already. When we try to pick a cut *C* for a node from *L*, we will check to see if some of the cut-set nodes (input nodes) of *C* are already LUT roots. If this is the case, *C* will not generate as many new LUT roots as other cuts do when those other cuts do not have this feature.[5] In other words, cut *C* takes advantage of existing resources and does not require new resources. This reduces the chances that a newly picked cut will cut into the interior territories of existing LUTs. As a result, the input nodes are shared among several mapped LUTs, and node duplications are reduced. When the ICS procedure is carried out, the current mapping iteration also considers the LUT roots generated from previous iterations (named as *tentative* LUT roots). Not all of the tentative LUT roots are considered. We only select those roots that have strong features, such as those that have a large number of fanouts. This indicates that these roots are most likely to be kept all the way to the final mapping.[6] The cost of cut *C* is recalculated and significantly reduced according to how many inputs it shares from existing LUT roots (both real and tentative). Thus, the cut selection is largely influenced by the local settings around the target node and its cuts.

Slack Distribution. We define the slack on a node *v* as follows:

$$Slack_v = Req_v - Arr_v \qquad (9)$$

When $Slack_v$ is greater than 0, it means *v* is not on the critical path so that there is more flexibility to choose a cut *C* that is not in X_v (refer to Section 4.1), as long as the required time propagated back to the input nodes of this cut is still larger or equal to the minimum arrival times on those nodes. It is easy to see that if *C* uses up all the slacks available on *v*, there will exist at least one path in F_v that will no longer have the flexibility to pick cuts outside of X_n, where *n* is on that path and $n \neq v$. So, we want to distribute the slacks along the edges of the entire paths to encourage more nodes on the paths to have the flexibility to search their solution space. We design a simple slack distribution technique, which is applied in terms of the adjusted cost. We define the slack of a cut *C* rooted on *v* as follows:

$$Slack_C = Req_v - 1 - \underset{i \in input(C)}{\text{MAX}} (Arr_i) \qquad (10)$$

If $Slack_C < 0$, *C* is not a *timing_feasible* cut. Choosing it will violate the optimal mapping depth constraint, so such a cut will be discarded. The larger the $Slack_C$, the better for *C* in terms of slack distribution effects. We then adjust the cost of *C* accordingly.

Cut Probing. This technique includes several heuristics that examine the characteristics of the target cut *C* from other angles and tries to probe the amount of area gain locally before making decisions about cut selection. We again use the example shown in Figure 3. First, cut C_3 has a nice feature that hasn't been

[5] Suppose a node *w* is a cut-set node for both cut C_a rooted on node *a* and cut C_b rooted on *b*. C_a is picked to implement LUT_a, and *w* is added into list *L* and becomes an LUT root. Later on, when we map *b*, C_b has an advantage because it shares input *w* with LUT_a.

[6] These tentative roots only influence cost adjustment and may or may not become the *real* LUT roots for the current iteration.

considered before. It will cause the fanout number of gate 3 to be reduced to 1 because C_3 covers both fanout nodes of gate 3. We thus reduce the cost of C_3 by a small constant because the number of total edges is reduced by using it. Second, considering the cut-set nodes and the root node of any cut C, these nodes themselves will all become LUT roots if C is picked, so they will have a small chance of being duplicated (due to the effect of our strong input sharing feature). If these nodes actually were duplicated in the previous iterations found by *update_profiling_info*, it means that cut C actually reduces duplications in the current iteration. The higher the number of duplicated cut-set nodes was, the more cost savings cut C gets. The third probing technique is related to the reconvergent paths. In Figure 3, there are two sets of reconvergent paths within cut C_2. This eliminates the chances of gate 1 and gate 3 getting duplicated even though they have multiple fanouts. If the previous iterations indicate that these gates were indeed duplicated before, we will have extra cost savings for choosing cut C_2. Notice this technique is different from the reconvergent-path consideration before in Section 4.2.

```
algorithm pick_cut
input: target node $v$, output: $LUT_v$.
if $v$ is critical, return the cut that provides $A_v$;
$best\_cost = \infty$;
for each timing_feasible cut $C$ on $v$ do
    $cost_C := A_C$;
    $share\_no$ := number of shared input nodes of $C$;
    if $share\_no = 0$ then
        $share\_no := 1$;
    else-if $share\_no = 1$ then
        $share\_no := 1.15$;
    end-if
    $cost_C := cost_C / share\_no$;
    $cost_C := cost_C - \varepsilon \cdot Slack_C$;
    $cost_C :=$ cut_probing($C$, $cost_C$);
    if ($best\_cost > cost_C$) then
        $best\_cost := cost_C$;
        $LUT_v := C$;
    end-if
end-for
output $LUT_v$;
```

Figure 5: Algorithm of *pick_cut*

All of the local adjustment techniques are implemented in the subroutine *pick_cut*. Its high-level description is shown in Figure 5. *share_no* is the number of shared inputs of a cut C with existing LUTs. The new cost is the old cost divided by *share_no*. When *share_no* is 1, we set it as 1.15 (obtained by empirical study) to count in the sharing effect. The direct effect of sharing an input with another LUT is that the portion of area from that input can be treated as ignorable for the target cut. Division achieves this purpose by weighing each shared input with the same amount of cost savings. It is a local operation concentrating on edge reduction. We use the term $\varepsilon \cdot Slack_C$ to count the slack distribution effect. Symbol ε is 0.3 in our case. The cost adjustment by cut probing is done in subroutine *cut_probing*.

Overall, local cost adjustment plays an important role in our mapping. We find that the combination of global cost estimation from cut enumeration and local cost adjustment from cut selection works greatly to reduce overall node duplications and save area. This combination is able to capture the optimization objective in a global fashion and yet maintain the best features of cuts according to its own virtues locally. In addition, ICS helps to weave everything together. It builds a new set of solution S based on the previously obtained intelligence, and is able to further improve the quality of results.

5. EXPERIMENTAL RESULTS

DAOmap is implemented using C language within the UCLA RASP system [25]. We first show the details of the comparison results between DAOmap and CutMap in terms of LUT counts and run-time using both MCNC benchmarks and industrial benchmarks. Then we will present some analysis for the effects of the various techniques we apply. This analysis helps us to better understand what factors are the most important to determine a good mapping solution under timing constraints. It also helps researchers in general to understand the impact of optimization techniques from both global and local optimization perspectives. In addition, it shows how learned intelligence can help to obtain better solutions under the iterative mapping paradigm.

5.1 Comparison Results with CutMap

Both DAOmap and CutMap are run on a 750 MHz SunBlade 1000 Solaris machine. We first collect the data by using the 20 largest MCNC benchmarks. Table 1 shows that DAOmap is 16.02% better than CutMap in terms of LUT counts on average, and runs 24.2X faster when both are mapped with 5-LUT. The mapping depth for each benchmark is the same for DAOmap and CutMap, so it is not shown. We also map with 4-LUT and 6-LUT. Using 4-LUT, DAOmap is 13.98% better on area and 13.2X faster. Using 6-LUT, DAOmap is 12.44% better on area and 4.7X faster. In this case, the run-time reduction is smaller because the number of cuts generated per node is relatively large when $K = 6$ (Section 3.2). As explained before, when K is large, cut-pruning can be applied to reduce the number of cuts, and this has proven to be very effective to reduce run-time in [6]. This will be considered in our future work. Nonetheless, when K is small, the number of cuts per node is a small constant in our algorithm. This offers a significant improvement compared to the complexity of CutMap. CutMap's worst complexity is $O(2Kmn^{\lfloor K/2 \rfloor+1})$ [21], where n is the number of the nodes in the network, and m is the number of edges in the network. CutMap uses a network flow computation to generate cuts. Although it does not need to generate all the cuts rooted on a node, the computation is considerably more expensive per single-cut generation when compared to that of cut enumeration. We would like to mention that CutMap can be run with a switch '–x', which will carry out depth relaxation on the non-critical paths. It can improve CutMap's mapping area but incur longer run-time. With this switch on, DAOmap is 11.62%, 14.76%, and 11.13% better on area, and 57.7X, 38.7X, and 10.1X faster on run-time than CutMap for 4-LUT, 5-LUT and 6-LUT mapping respectively. Notice that we show the results directly after mapping for both algorithms. People can run a postprocessing procedure, such as a predecessor packing algorithm after the initial mapping. We try a powerful packing procedure, *mpack* (available in the RASP package). It can further improve CutMap's area results by 3-5% on average for various K values. Interestingly, it does not improve DAOmap's results much - only 1% on average. Since packing is not the research goal of this work, we omit the detailed data here.

We also run DAOmap and CutMap on six large industrial benchmarks, which are provided to us under non-disclosure agreements. Table 2 shows the results (no –x switch for CutMap and no packing for both). For two of the benchmarks, CutMap cannot finish mapping after 10 hours. We can observe that DAOmap obtains better results consistently. Especially, the run-time improvement is close to two orders even without counting the two cases where CutMap drags on. This shows the efficiency of the cut-enumeration-based algorithm and its good scaling feature toward million-gate FPGA designs.

	CutMap		**DAOmap**		**Comparison**	
Bench marks	LUT No.	Run Time (s)	LUT No.	Run Time (s)	LUT (Reduce)	Run Time (Improve)
alu4	1233	17	1065	2	-13.6%	8.5
apex2	1578	78	1352	2	-14.3%	39.0
apex4	1153	59	931	2	-19.3%	29.5
bigkey	1465	8	1245	2	-15.0%	4.0
clma	6345	888	5405	25	-14.8%	35.5
des	1075	6	965	2	-10.2%	3.0
diffeq	990	14	817	2	-17.5%	7.0
dsip	908	4	686	1	-24.4%	4.0
elliptic	2556	102	1965	8	-23.1%	12.8
ex1010	4181	395	3567	14	-14.7%	28.2
ex5p	870	125	778	2	-10.6%	62.5
frisc	2398	242	1999	8	-16.6%	30.3
misex3	1141	33	980	2	-14.1%	16.5
pdc	3757	288	3222	15	-14.2%	19.2
s298	1649	941	1257	7	-23.8%	134.4
s38417	4437	248	3819	16	-13.9%	15.5
s38584	3757	27	2982	79	-20.6%	0.3
seq	1414	34	1188	2	-16.0%	17.0
spla	3278	154	2734	13	-16.6%	11.8
tseng	759	5	706	1	-7.0%	5.0
Ave.					**-16.02%**	**24.2X**

Table 1: LUT number and run-time comparisons of DAOmap vs. CutMap using 5-LUT on the 20 largest MCNC benchmarks

5.2 Impact of Various Techniques

To show the effectiveness of our mapping algorithm and provide some insights on the most important factors for area minimization, we carry out experiments to evaluate individual techniques presented in this paper. Table 3 shows the results.[7] Each technique is taken out of the mapping procedure individually to assess how the quality of result drops compared to the flow where every technique is included. The bigger the percentage drops, the more effective the technique is for area reduction.

The important factors are input sharing, min-cost propagation, global cost adjustment, and ICS. Input sharing proves to be the most important technique to reduce area because it reduces the number of edges and node duplications significantly. The min-cost propagation is trying to evaluate how accurate our cost estimation model is in general. Formula (4) shows that A_v is the cost of the cut that has the minimum cost among all the cuts in X_v.

[7] The percentages in the table and figure in this section are based on the mapping results using 5-LUT across the 20 largest MCNC benchmarks.

	CutMap		**DAOmap**		**Comparison**	
Bench marks	LUT No.	Run Time (s)	LUT No.	Run Time (s)	LUT (Reduce)	Run Time (Improve)
big1	9928	301	9169	93	-7.6%	3.2
big2	-	>10H.	14625	708	-	-
big3	10005	28926	9031	106	-9.7%	272.9
big4	11800	583	9364	156	-20.6%	3.7
big5	-	>10H.	32230	3377	-	-
big6	39000	14437	32028	402	-17.9%	35.9
Ave.					**-13.98%**	**78.9X**

Table 2: DAOmap vs. CutMap on large industrial benchmarks using 5-LUT

If we remove this operation, and A_v is just the cost from the first cut that offers the minimum arrival time Arr_v, and we keep everything else the same, we observe that the quality of result drops by 4.35%. This indicates that our cost estimation through min-cost propagation is working well overall, so it has a large influence on the mapping result. Global duplication cost adjustment offers the next largest gain. It shows how important it is to estimate the additional cost due to potential node duplications. The globally adjusted cost is closer to reality and makes a positive impact for cut selection. ICS offers considerable gain and shows the effectiveness of the concept of iterative improvement. Slack distribution offers some observable help.

The cost function evaluation is done against a flow that treats the cost of each cut as $U_C = 1$. Surprisingly, our sophisticated cost function does not offer much help (only 0.27%). This is counter-intuitive because we show that a cut can be completely different in every aspect from another cut. We have tried other cost functions, and the impact is similar if not worse. Since the area is propagated from PIs, we believe our area model naturally prefers cuts that cover more nodes because the overall estimated area will be smaller that way even when we count each cut as having one unit area. Nonetheless, our cost function considers other factors and tries to augment the good features of the area estimation model. We believe this cost function offers valuable insights for technology mapping in general. Cut probing also provides a limited gain. We think its gain may be shadowed by the strong influences of other techniques because cut probing usually does not change the cost of a cut significantly. Cut probing itself offers a neat way to take advantage of learned intelligence and the characteristics of cut-enumeration-based method. The idea can be extended into other optimization fields that use cut-enumeration technique as a framework, such as pattern generation that is applied in many VLSI optimization areas.

Techniques	**% dropped**
Cut Enumeration	
Min-cost propagation	4.35%
Global cost adjustment	2.68%
Cut Selection	
Input sharing	4.55%
Iterative cut selection (ICS)	2.04%
Slack distribution	0.86%
Cut probing	0.25%
Cost Function	0.27%

Table 3: Individual technique evaluation

Next, we study how ICS influences mapping results alone. Figure 6 shows the details. For mapping with a single iteration only (the base case), we first go through the netlist and set all the nodes whose fanout numbers are greater than 3, as tentative LUT roots (like a manual profiling used in [19]). Then we go through the single iteration of the cut selection procedure. There is no other profiling information involved. For other iteration numbers, the full-blown ICS procedure is activated. The interesting observation is that when the iteration number is more than 3, ICS actually starts to hurt the final mapping results slightly. We believe this is because that the profiling procedure accumulatively stores information through the iterations, and this eventually exceeds what the current iteration can take advantage of. In other words, the intelligence learned previously starts to become vague. The iteration number of 3 offers the best result, and it is used in all of the DAOmap results.

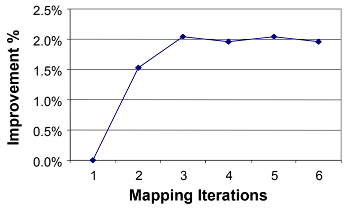

Figure 6: Area improvement along mapping iterations

We would like to emphasize that we also evaluate the individual techniques using the industrial benchmarks. Similar conclusions are drawn as those shown in Table 3. This indicates that our area optimization techniques are general enough to target circuits with different sizes and various characteristics.

6. CONCLUSION

In this paper we presented a technology mapping algorithm, DAOmap, for FPGA architectures to minimize chip area under timing constraints. Our approach closely monitored various node duplication scenarios during both the cut-enumeration stage and the cut-selection stage. We designed novel heuristics that captured the mapping cost accurately with consideration of both local and global optimization information. Our work frame offered more flexibility and excellent efficiency for FPGA area optimization and guaranteed optimal mapping depth under the unit delay model. We evaluated these novel techniques in a systematic way and provided insights on the most important factors for technology mapping under the cut-enumeration paradigm. Experimental results showed that DAOmap produced significant quality and run-time improvements. Compared to the state-of-the-art depth-optimal, area minimization mapping algorithm CutMap, DAOmap is 16.02% better on area and runs 24.2X faster on average when both algorithms mapped to FPGAs using 5-LUT. Our future work will add cut-pruning techniques for larger *K* values and evaluate how much this will influence mapping results and run-time.

Acknowledgement

This work is partially supported by the California MICRO program and the NSF Grant CCR-0306682.

REFERENCES

[1] R. J. Francis, et al., "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *DAC*, 1991.

[2] R. Murgai, et al., "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *ICCAD*, Nov., 1991.

[3] K. Karplus, "Xmap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays," *DAC*, 1991.

[4] N.-S. Woo, "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," *DAC*, 1991.

[5] P. Sawkar and D. Thomas, "Technology Mapping for Table-Look-Up Based Field Programmable Gate Arrays," *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, Feb. 1992.

[6] J. Cong, C. Wu, and E. Ding, "Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution," *FPGA,* Feb. 1999.

[7] A. Farrahi and M. Sarrafzadeh, "Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping," *IEEE TCAD,* Vol. 13, No. 11, pp. 1319-1332, Nov. 1994.

[8] R. J. Francis, J. Rose, and Z. Vranesic, "Technology mapping for lookup table-based FPGA's for performance," *ICCAD*, Nov. 1991.

[9] R. Murgai, et al., "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," *ICCAD,* Nov., 1991.

[10] K. C. Chen, et al., "DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization," *IEEE Design and Test of Computers*, vol. 9, no. 3, pp. 7-20, Sep., 1992.

[11] J. Cong and Y. Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *ICCAD*, Nov. 1992.

[12] H. Yang and D. F. Wong, "Edge-map: Optimal Performance Driven Technology Mapping for Iterative LUT based FPGA Designs," *ICCAD*, Nov. 1994.

[13] P. Pan and C.L. Liu, "Optimal Clock Period FPGA Technology Mapping for Sequential Circuits," *DAC*, June 1996.

[14] A.H. Farrahi and M. Sarrafzadeh, "FPGA Technology Mapping for Power Minimization," *Proc. of Intl. Workshop in Field Programmable Logic and Applications*, 1994.

[15] J. Anderson and F. N. Najm, "Power-Aware Technology Mapping for LUT-Based FPGAs," *IEEE Intl. Conf. on Field-Programmable Technology*, 2002.

[16] Z-H. Wang et al., "Power Minimization in LUT-Based FPGA Technology Mapping," *ASPDAC*, 2001.

[17] H. Li, W. Mak, and S. Katkoori, "Efficient LUT-Based FPGA Technology Mapping for Power Minimization," *ASPDAC*, 2003.

[18] J. Lamoureux and S.J.E. Wilton, "On the Interaction between Power-Aware CAD Algorithms for FPGAs," IEEE/ACM International Conference on Computer Aided Design, 2003.

[19] D. Chen, et al., "Low-Power Technology Mapping for FPGA Architectures with Dual Supply Voltages," *FPGA*, Feb. 2004.

[20] J. Cong and Y. Ding, "On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping," *IEEE Trans. on VLSI Systems*, vol. 2, no. 2, pp. 137-148, June 1994.

[21] J. Cong and Y. Hwang, "Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping," *FPGA*, Feb. 1995.

[22] C. Legl, B. Wurth, and K. Eckl, "A Boolean Approach to Performance-Directed Technology Mapping for LUT-Based FPGA Designs," *DAC*, June 1996.

[23] J. Cong and Y. Ding, "Beyond the Combinatorial Limit in Depth Minimization for LUT-Based FPGA Designs," *ICCAD*, Nov. 1993.

[24] S-C Chang, M. Marek-Sodowska, and T. Hwang, "Technology Mapping for TLU FPGA Based on Decomposition of Binary Decision Diagrams," *IEEE Transactions on CAD*, Vol. 15, No. 10, pp. 1226-1236, Oct. 1996.

[25] UCLA RASP FPGA/CPLD Technology Mapping & Synthesis Package, http://ballade.cs.ucla.edu/software_release/rasp/htdocs/ .