Simultaneous Depth and Area Minimization in LUT-based FPGA Mapping

Jason Cong and Yean-Yow Hwang

Department of Computer Science
University of California, Los Angeles, CA 90024

January 31, 1995

Abstract

In this paper, we present an improvement of the FlowMap algorithm, named CutMap, which combines depth and area minimization in the mapping process by computing min-cost min-height K-feasible cuts for nodes which are on critical paths for depth minimization and computing min-cost K-feasible cuts for nodes which are not on any critical path for area minimization. CutMap guarantees depth-optimal mapping solutions in polynomial time as the FlowMap algorithm but uses considerably fewer LUTs. We have implemented CutMap and tested it on the MCNC logic synthesis benchmarks. For depth-optimal mapping solutions, CutMap uses 20% fewer K-LUTs than FlowMap without post-processing, and uses 13% fewer K-LUTs than FlowMap when post-processing operations for area minimization are applied to both solutions. When targeting for Xilinx X3000 FPGA family, CutMap uses 11% fewer CLBs than FlowMap. We also tested CutMap followed by the depth relaxation routines in FlowMap_r algorithm, which achieves area minimization by depth relaxation. CutMap followed FlowMap_r performs better than FlowMap_r.

1. Introduction

For VLSI ASIC design and system prototyping, the field programmable gate array (FPGA) is a popular technology due to its short design cycle and low manufacturing cost. There are two major types of FPGA, the multiplexer based FPGA (e.g. Actel) and the lookup table (LUT) based FPGA [Xi93b]. The architecture of LUT-based FPGA is an array of configurable logic blocks (CLB), each has K input lines, 2^K bit internal memory and one output line (some architecture has two outputs). Such a device can implement any Boolean function of up to K variables. The LUT-based FPGA technology mapping problem is to convert a Boolean network into a functionally equivalent LUT network.

Previous work on LUT-based FPGA technology mapping aims at either area minimization, or depth minimization, or routability optimization as their primary goal. Mappers such as Chortle [FrRC90], Mis-pga [MuNS90], Xmap [Ka91a], FGMap [LaPV93] focus on minimizing the number of LUTs. Mappers such as Chortle-d [FrRV91b], Mis-pga-delay [MuSB91], DAG-Map [ChCD92], FlowMap [CoDi94a] focus on minimizing the delay of the LUT network. Other mappers, such as Rmap [ScKC92] optimize the routability of the mapping solution.

Delay minimization has been achieved through various approaches. Chortle-d algorithm minimizes depth of LUT network using optimal tree based mapping algorithm and bin packing procedure. Mis-pga-delay combines technology mapping with layout synthesis for delay minimization. DAG-Map uses Lawler's labeling algorithm for depth minimization. An important advance in LUT-based FPGA depth minimization is the FlowMap algorithm [CoDi94a] which guarantees depth-optimal mapping solutions in polynomial time for general K-bounded networks. On average, FlowMap outperforms Chortle-d by 4.8% in depth and 50.4% in area, DAG-Map by 2.4% in depth and 8.6% in area, and Mis-pga-delay by 7.1% in depth and 9.8% in area [CoDi94a]. However, one limitation of the FlowMap algorithm is that area minimization is not considered in the mapping process, but achieved by a separate sequence of post-processing operations, including gate-decomposition, predecessor-packing [ChCD92] and FlowPack [CoDi94a]. These post-processing operations, although effective, may lead to sub-optimal solutions in terms of area minimization.

In this paper, we present an improvement of the FlowMap algorithm, named CutMap, which combines depth and area minimization in the mapping process by computing min-cost min-height K-feasible cuts for nodes which are on critical paths for depth minimization and computing min-cost K-feasible cuts for nodes which are not on any critical path for area minimization. The min-cost K-feasible cut computation encourages sharing for LUTs when mapping different parts of the circuits. CutMap still guarantees depth-optimal mapping solutions in polynomial time but uses considerably fewer LUTs. We have implemented CutMap and tested it on the MCNC logic synthesis benchmarks. For depth-optimal mapping solutions, CutMap uses 20% fewer K-LUTs than FlowMap without post-processing, and uses 13% fewer K-LUTs than

FlowMap when post-processing operations for area minimization are applied to both solutions. When targeting for Xilinx X3000 FPGA family, CutMap uses 11% fewer CLBs than FlowMap. In addition, we tested CutMap followed by depth relaxation routines in FlowMap_r [CoDi94b] for further area minimization. CutMap followed FlowMap_r also performs better than FlowMap_r in term of number of LUTs in the mapping solution.

The remainder of this paper is organized as follows. Section 2 gives the problem formulation and defines the basic terminology. Section 3 reviews FlowMap algorithm and present our CutMap algorithm. Experimental results are presented in Section 4. Section 5 gives conclusion and future research direction.

2. Problem Formulation

We use the notions defined in [CoDi94b]. A combinational Boolean network can be represented by a directed acyclic graph (DAG) where a node represents a logic gate and a directed edge (u,v) represents a connection from the output of gate u to the input of gate v. A primary input (PI) node is a node of in-degree zero and a primary output (PO) node is a node with no outgoing edge. Other nodes are called internal nodes. The depth of a node v is the number of edges on the longest path from any PI to v. A PI node has a depth of zero. The depth of a network is the largest node depth among POs. Let $input(v)$ denote the set of nodes which are fanins of node v and $output(v)$ denote the set of nodes which are fanouts of node v. Given a subgraph H of a Boolean network, let $input(H)$ represent the set of distinct nodes outside H which supply inputs to the gates in H. A network is K-bounded if $|input(v)| \le K$ for every v in the network.

A cone at v, denoted as C_v, is a subgraph of logic gates (excluding PIs) consisting of v and its predecessors such that every path connecting a node in C_v and v lies entirely in C_v. Node v is said the *root* of C_v. The fanin cone at v is the cone at v of largest size. A cone C_v is K-feasible if $input(C_v) \le K$. A K-LUT is a LUT with K inputs. A K-feasible cone can be implemented by a K-LUT. When a K-LUT LUT_v implements a cone C_v, we say LUT_v *covers* C_v or LUT_v implements v. In this paper, we study the following problem.

Bounded-Depth Min-Area Mapping Problem (BDMAM Problem): Given a K-bounded network and a depth bound D, cover the entire network using K-LUTs to form a functionally equivalent LUT-based network of depth no larger than D and use as few K-LUTs as possible.

It has been shown recently that the area-optimal technology mapping problem for K-bounded networks is NP-complete [FaSa94]. The BDMAM problem is a more general problem because the solution is additionally constrained by a depth bound D. When D is sufficiently large, the BDMAM problem becomes the area-optimal mapping problem. Hence, the BDMAM problem is NP-hard. We shall develop efficient heuristic algorithm for the BDMAM problem.

We introduce some concepts of cuts in the rest of this section. Given a network $N = (V(N), E(N))$ with a source s and a sink t, a cut $(X, \overline{X})$ is a partition of $V(N)$ such that $s \in X$ and $t \in \overline{X}$. For a given cut $(X, \overline{X})$, the edge set of the cut, denoted $e(X, \overline{X})$, is defined as $e(X, \overline{X}) = \{ (u,v) \mid (u,v) \in E(N), u \in X, v \in \overline{X} \}$ and the node set of the cut, denoted $n(X, \overline{X})$, is defined as $n(X, \overline{X}) = \{ u \mid (u,v) \in e(X, \overline{X}) \}$. The *edge cut-size* and the *node cut-size* of a cut is the cardinality of its edge set and node set, respectively. For a given network with a source and a sink, a *min-edge cut* is a cut of minimum edge cut-size and a *min-node cut* is a cut of minimum node cut-size. To compute a min-edge cut for a network, one can assign unit capacity to every edge and run a max-flow algorithm. To compute a min-node cut, one can assign infinite capacity to every edge, divide each node into two nodes which are linked by a directed edge of unit capacity, and run a max-flow algorithm. The edge set of the modified network corresponds to the node set of the min-node cut in the original network. Figure 1(a) shows a combinational network where a, b, c are primary inputs and f is a primary output (and sink). A source node s together with edges from s to primary inputs are created (in dashed lines) for the purpose of flow computation. Figure 1(b) shows a network modified for min-node cut computation. The min-edge cut of Figure 1(b), which is $(\{ s, a, a', b, b', c, c', d, e \}, \{ d', e', f \})$, corresponds to the min-node cut of Figure 1(a), which is $(\{ a, b, c, d, e \}, \{ f \})$. For LUT-based technology mapping, it is natural to consider node set instead of edge set of a cut. For the rest of the paper, we refer min-cut as min-node cut, cut set as node set, and cut-size as node cut-size.

A cut $(X, \overline{X})$ is K-feasible if its cut-size is not larger than K. To bias for certain nodes in a cut set, we assign a cost of either zero or one to every node in the network. A node is a preferred node if it has a cost of zero. In the CutMap algorithm, the preferred nodes are those which have been implemented or likely to be implemented using K-LUTs. The cost of a cut is the summation of the cost of nodes in the cut set. A *min-cost cut* is a cut of minimum cost. A min-cut is not

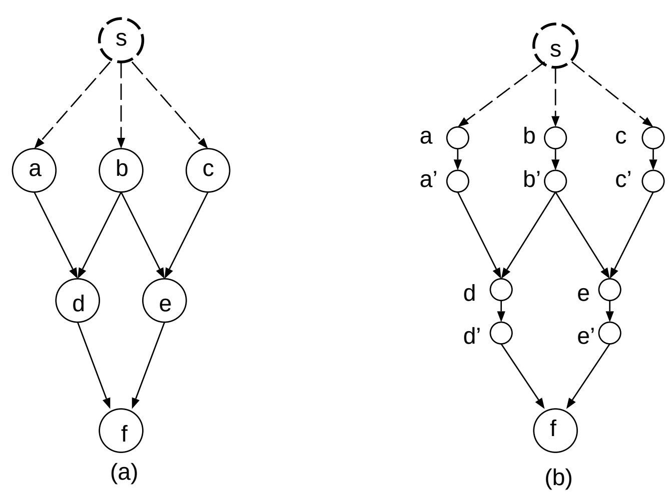


Figure 1 (a) a network (b) a modified network for node cut necessary a min-cost cut. For example, if node a,b,c have a cost of zero and node d,e have a cost of one in Figure 1, then the min-cost cut for f as a sink is $(\{a,b,c\},\{d,e,f\})$ while the min-cut is $(\{a,b,c,d,e\},\{f\})$. A min-cost cut may not always be K-feasible. In general, we are interested in the min-cost K-feasible cut because of its low cost and implementability by a K-LUT.

3. CutMap Algorithm

In this section, we discuss the CutMap algorithm, the optimal min-cost K-feasible cut algorithm, as well as the speed-up method and a heuristic for the min-cost K-feasible cut algorithm. First, we briefly review the FlowMap algorithm, which is the basis of the CutMap algorithm.

3.1. Review of FlowMap Algorithm

FlowMap is a LUT-based FPGA mapper which produces depth-optimal mapping solution for K-bounded Boolean networks. Given a K-bounded Boolean network N, let N_v denote the subnetwork consisting of node v and all the predecessors of v. The label of v, denoted $label(v)$, is defined as the depth of the optimal K-LUT mapping solution of N_v. Let $D_{\mathrm{opt}}(N)$ denotes the largest $label(v)$ for all $v \in V(N)$, i.e., $D_{\mathrm{opt}}(N) = \max\{\, label(u) \mid u \in V(N)\}$. In the first phase, FlowMap calculates a label for every node in topological sorting order. Consider the computation of $label(v)$ for a node v in the sorting list. Let p be the largest label of nodes in N_v, excluding v. Then node v has a label of either p or $p+1$ [CoDi94a]. When node u is a fanin node of node v, *collapsing* node u into v denotes the operation of removing u from N_v and replacing every edge $(t,u) \in E(N_v)$ by (t,v). To determine the label for v, FlowMap collapses all nodes u with $label(u) = p$ into v and computes a min-cut in N_v. If the min-cut is K-feasible, v has a label of p (since every node in the cut-set has a label of $p-1$ or less) and the K-feasible min-cut is stored at v. Otherwise v has a label of $p+1$ and the cut $(V(N_v)-\{v\},\{v\})$ is stored at v. In either case, we said a *min-height* cut is calculated for node v. Clearly, all stored cuts are min-height K-feasible min-cuts.

In the second phase, FlowMap generates a mapping solution. Let $(X_v,\bar{X}_v)$ be the cut stored at v in the first phase and support of v, denoted $sp(v)$, be the cut set of $(X_v,\bar{X}_v)$, i.e. $sp(v) = n(X_v,\bar{X}_v)$. Initially, let Q be the set of all PO nodes. FlowMap repeats the following three steps in the second phase until Q contains only PI nodes. (1) Remove a node v from Q. (2) Implement v by a K-LUT covering $\bar{X}_v$. And (3) add nodes in $sp(v)$ to Q. A number of post-processing routines (gate-decomposition, predecessor-packing [ChCD92] and FlowPack [CoDi94a]) are carried out to decrease the number of LUTs. FlowMap's mapping solution has the property that every LUT LUT_v on v has the minimum depth equal to $label(v)$. In fact, this is unnecessary for those LUTs which are not on critical paths in order to have a depth-optimal mapping solution. Applying depth relaxation operations and re-mapping can further decrease the number of LUTs in the FlowMap mapping solution as reported in FlowMap_r [CoDi94b].

3.2. Overview of CutMap Algorithm

In our approach, we do not store the min-cut for each node in the first phase. Instead, we compute either a min-cost min-height K-feasible cut or a min-cost K-feasible cut depending on the depth criticality of the mapped node in the second phase. We define following attributes to measure the depth criticality of a node v:

$label(v)$	depth of LUT_v in the mapping solution
$latest(v)$	largest depth of LUT_v in order to have depth-optimal mapping solution
$slack(v)$	$latest(v) - label(v)$

A node v is on the critical path if $slack(v) = 0$, i.e., $label(v) = latest(v)$. In the CutMap algorithm, $label(v)$ can only increase, $latest(v)$ can only decrease, and $label(v) \le latest(v)$ always holds. Whenever a cut is computed in the CutMap algorithm, these attributes are updated to reflect most current depth criticality.

CutMap algorithm has two phases. In the first phase, CutMap calculates $label(v)$ using FlowMap algorithm for each node v. The optimal depth $D_{\text{opt}}(N)$ is then computed. If $D_{\text{opt}}(N) > D$, the given depth bound, CutMap stops and claims that no solution exists with respect to the given depth bound. Otherwise, CutMap proceeds to assign $latest(v) = D_{\text{opt}}(N)$ for every $v \in V(N)$. In the second phase, CutMap initializes Q to be the set of all POs and repeats the following steps until Q contains only PIs. (1) Remove the node v of minimal slack in Q from Q. (2) Assign a cost to every node in N_v based on the likelihood of implementing the node (to be discussed in Section 3.3). (3) Compute a proper K-feasible cut depending on $slack(v)$. If $slack(v) = 0$, CutMap collapses all nodes u which has $label(u) \ge label(v)$ in N_v and computes a min-cost K-feasible cut. The resulting cut is a min-cost min-height K-feasible cut. If $slack(v) > 0$, no node-collapsing is performed and CutMap computes a min-cost K-feasible cut. In either case, an optimal min-cost K-feasible cut algorithm is used and will be discussed in detail in Section 3.4. And (4) add $sp(v)$ to Q. A pseudo code for the CutMap algorithm is shown in Figure 2.

If a cut has been computed for a mapped node v, then for every nodes $u \in sp(v)$, we update $latest(u)$ according to the following rule: $latest(u)$ = min($latest(u)$, $latest(v) - 1$). In FlowMap algorithm, we always have $label(v) \ge label(u) + 1$ for each $u \in sp(v)$. However, in CutMap algorithm, since we no longer compute a min-height K-feasible cut for v when $slack(v) > 0$, it might happen that for some node $u \in sp(v)$ we have $label(v) = label(u)$. In this case, $label(v)$ will be increased by one. In addition, we need to recompute label for every node t which is a transitive successor of v, since v might be in the cut set of t when the CutMap implements node t later on. However, we would like to point out that CutMap still returns a depth-optimal mapping solution if the given depth $D \ge D_{\text{opt}}(N)$ since node v is not on any critical path in this case. CutMap algorithm takes $O(LM)$ time to return a solution where L is the number of LUTs in the solution (bounded by the number of nodes in N) and M is the time required by each execution of

CutMap: Algorithm for the BDMAM Problem

Procedure CutMap(N, K, D,**var** T)

/* Given a Boolean network N, a LUT input size K, and a depth bound D,
return a functionally equivalent K-LUT network T */

Phase 1

```
for each node v ∈ N do
        label(v) = compute_label(v) by FlowMap algorithm;
D_opt = max({ label(v) | v ∈ N });
if D_opt > D return no solution;
for each node v ∈ N do latest(v) = D_opt;
```

Phase 2

```
Q = { v | v is a PO};
while Q contains PO or internal node do
        v = extract_min_slack_node(Q);
        assign cost to nodes in N_v;
        if slack(v) = 0
                (X_v, X̄_v) = Min_Cost_Min_Height_K_Feasible_Cut(N_v, v, K);
        else if slack(v) > 0
                (X_v, X̄_v) = Min_Cost_K_Feasible_Cut(N_v, v, K);
        for each u ∈ n(X_v, X̄_v) do
                latest(u) = min(latest(u), latest(v) − 1);
        if ∃u ∈ n(X_v, X̄_v) such that label(u) = label(v)
                label(v) = label(v) + 1;
                for each t which is a successor of v do
                        label(t) = compute_label(t) by FlowMap algorithm;
        end if
        Q = Q ∪ n(X_v, X̄_v) − { v };
end while

form LUT network T;
return T;
```

Figure 2 CutMap algorithm

min-cost K-feasible cut procedure (discussed in Section 3.4).

3.3. Predictive Cost Assignment

A min-cost K-feasible cut is computed for every node $v \in Q$ in the CutMap algorithm. Before the cut computation for v, a node in N_v is assigned a cost of zero if it has been implemented or likely to be implemented later on. Otherwise, it is assigned a cost of one (in this case, we have to introduce a new LUT if we include this node in the cut). The cost assignment is an important decision that CutMap has to make in order to achieve the goal of area minimization. At any time, a node u is assigned a cost of zero if u has been implemented. Primary input nodes are always assigned a cost of zero since there is no need to implement them. Primary output nodes are also assigned a cost of zero because they must be implemented anyway.

Clearly, the population of zero-cost nodes increases as the mapping process proceeds since more and more nodes are implemented. However, at the beginning of phase two, there are very few zero-cost nodes (only PI and PO nodes). Hence, a min-cost cut is a min-cut in most cases. In order to influence the min-cost cut procedure to choose the ‘‘appropriate’’ nodes in the cut, we use the MFFC concept to predict which nodes are likely to be implemented later on. A fanout-free cone (FFC) at v, denoted FFC_v, is a cone of v such that for every $u \neq v \in FFC_v$, $output(u) \subseteq FFC_v$. The MFFC at v, denoted $MFFC_v$, is the fanout-free cone at v of maximal number of nodes [CoDi94b]. The MFFCs have the following properties: (i) If $w \in MFFC_v$, then $MFFC_w \subseteq MFFC_v$. (ii) Two MFFCs are either disjoint or one contains another. (iii) Let $(X_v, \bar{X}_v)$ be a min-cut of N_v. Then for each $MFFC_i$ in the MFFC decomposition of N_v, either $X_v \cap MFFC_i = \emptyset$ or $MFFC_i \subseteq X_v$ [CoDi94b, CoLB94]. We decompose the network into MFFCs. Such decomposition is unique. If the size of $MFFC_v$ is sufficiently large, node v is very likely to be implemented by a LUT. Otherwise, a K-feasible cut through $MFFC_v$ will force more nodes to be implemented. Hence, we assign roots of large MFFC’s a cost of zero. Another advantage of such assignment is based on the fact that roots of MFFC are multiple fanout nodes. It provides opportunity for sharing the LUTs and leads to better area minimization result. We observed that when MFFC-root based cost assignment method is used, CutMap produces consistently better results. In our implementation, roots of MFFCs which contain five or more nodes are assigned a cost of zero.

3.4. Min Cost K-Feasible Cut

In the second phase of the CutMap algorithm, a min-cost K-feasible cut is calculated for each node v in the queue Q. In this section, we present an efficient optimal min-cost K-feasible cut algorithm.

3.4.1. Optimal Min Cost K-Feasible Cut

Let N_v be a Boolean network of single primary output v. Assume all nodes in the network have been assigned a cost of either zero or one, according to the principle discussed in the previous section. Our goal is to find a K-feasible cut with sink v such that the cost of the cut is minimum. A trivial approach enumerates all cuts of size no more than K and selects the one of the minimum cost. But it takes $[C(n,2) + \cdots + C(n,K)] \cdot O(m) = O(mn^{K+1})$ time where n is the number of nodes in N_v, m is the number of edges in N_v, and $C(n,K)$ is the number of K-

combinations from n numbers. In the remainder of this section, we present a more efficient optimal algorithm which takes $2 \cdot [C(n, 1) + \cdots + C(n, \lfloor K/2 \rfloor)] \cdot O(Km) = O(2Kmn^{\lfloor K/2 \rfloor + 1})$ time. For $n = 1000$, $K = 5$, and $m = 3n$ (i.e., average fanout of each node is 3), $mn^{K+1} = 3 \cdot 10^{21}$ while $2Kmn^{\lfloor K/2 \rfloor + 1} = 3 \cdot 10^{13}$. Therefore, our algorithm is significantly faster than the trivial method.

Let V_0 and V_1 be the sets of nodes of cost zero and one, respectively. Our strategy is to search the cut from low cost end toward high cost end. Suppose there exists a cut $(X_v, \bar{X}_v)$ of cost zero. It must be the case that $n(X_v, \bar{X}_v) \cap V_1 = \emptyset$. To check for this case, we define capacity for every node as follows: $capacity(u) = \infty$ if $u \in V_1$ and $capacity(u) = 1$ if $u \in V_0$. Since PI nodes have a cost of zero, this capacity assignment guarantees a finite maximal flow of value no more than the number of PIs. In this case, it is easy to see that a K-feasible zero-cost cut exists if and only if the flow is not larger than K.

When there does not exist a zero-cost K-feasible cut, we proceed to search for a cut of unit-cost. Suppose there exists a K-feasible cut $(X_v, \bar{X}_v)$ of cost one, it must be the case that $|n(X_v, \bar{X}_v) \cap V_1| = 1$. In our algorithm, we repeatedly pick a node u out from V_1, add u to V_0, and test for a zero-cost cut in the resulting network $N_v(u)$. Clearly, a zero-cost cut in $N_v(u)$ including u corresponds to a unit-cost cut in N_v. If the zero-cost cut in $N_v(u)$ is not K-feasible, u is put back to V_1 and another node $u' \in V_1$ is chosen to form $N_v(u')$ for further testing. This process terminates when a unit-cost cut is found or every node in V_1 has been tested. In general, for a cost c, we repeatedly choose a subset S of V_1 such that $|S| = c$, assign nodes in S a zero-cost to form a new network $N_v(S)$ and test if $N_v(S)$ has a zero-cost K-feasible cut including S. It is easy to see that $N_v(S)$ has a zero-cost cut including S if and only if N_v has a cut of cost c with S in the cut set.

When cost c exceeds $\lfloor K/2 \rfloor$, we assign cost in a different way to reduce the computational complexity. If a K-feasible cut $(X_v, \bar{X}_v)$ has a cost c, it must be the case that $|n(X_v, \bar{X}_v) \cap V_0| \leq K - c$. Hence, we choose a set S consisting of $K - c$ zero-cost nodes, set their node capacities to zero (i.e., to force them to be in the cut set), set the capacities of remaining zero-cost nodes to infinity, set the capacities of unit-cost nodes to one and denote the resulting network $N_v(S)$. Then we compute a max-flow in $N_v(S)$. It is easy to see that a K-feasible cut of cost c exists in N_v if and only if the max-flow in $N_v(S)$ has a value c. If the max-flow in $N_v(S)$ is larger than c, another set S of size c will be tried, until all possible S sets have been tested. It can be shown the max-flow in $N_v(S)$ is at least c. Otherwise, a K-feasible cut of cost smaller than c should have been discovered in earlier steps. Using this method, we reduce the number of max-flow computation from $C(n_1, c)$ to $C(n_0, K - c)$ for testing the existence of K-feasible cut of cost c, where $n_0 = |V_0|$ and $n_1 = |V_1|$, both of them are bounded by n. This method limits the growth of computational complexity. When $c = K$, we simply need a single min-cut computation. Because N_v is K-bounded, a K-feasible cut must exist. When no cut of cost $K - 1$ or less is found, the min-cut must be the min-cost cut. Figure 3 gives the pseudo code for the optimal min-cost

```
Procedure Min_Cost_K_Feasible_Cut(N_v, v, K)

V_0 = { u | cost(u) = 0 }, V_1 = { u | cost(u) = 1 };

for c = 0 to ⌊K/2⌋ do
    for each subset S of V_1 such that |S| = c do
        V_1 = V_1 − S, V_0 = V_0 ∪ S;
        capacity(u) = 1 for all u ∈ V_0;
        capacity(u) = ∞ for all u ∈ V_1;
        f_c(S) = Max_Flow(N_v);
        if value(f_c(S)) ≤ K
            a cut of cost c is found, return the cut;
        V_1 = V_1 ∪ S, V_0 = V_0 − S;
    end for
for c = ⌊K/2⌋ + 1 to K − 1 do
    for each subset S of V_0 such that |S| = K − c do
        V_0 = V_0 − S;
        capacity(u) = 0 for all u ∈ S;
        capacity(u) = ∞ for all u ∈ V_0;
        capacity(u) = 1 for all u ∈ V_1;
        f_c(S) = Max_Flow(N_v);
        if value(f_c(S)) ≤ c
            a cut of cost c is found, return the cut;
        V_0 = V_0 ∪ S;
    end for
for c = K to K do
    return a min-cut;
```

Figure 3 Optimal Min Cost K-feasible Cut Algorithm

K-feasible cut algorithm. We have the following result:

Theorem 1 The min-cost-K-feasible-cut algorithm computes an optimal min-cost K-feasible cut in a K-bounded network N_v in $O(2Kmn^{\lfloor K/2 \rfloor + 1})$ time, where n is the number of nodes in N_v and m is the number of edges in N_v.

Proof Let $n_1 = |V_1|$ and $n_0 = |V_0|$. For each $c \in [0, \lfloor K/2 \rfloor]$, the Max_Flow procedure will be executed at most $C(n_1, c)$ times. For each $c \in [\lfloor K/2 \rfloor + 1, K-1]$, the Max_Flow procedure will be executed at most $C(n_0, K-c)$ times, i.e., $C(n_0, \lfloor K/2 \rfloor)$ times if K is odd and $C(n_0, \lfloor K/2 \rfloor - 1)$ times if K is even. For $c = K$, Max_Flow procedure is executed exactly once. Totally, the Max_Flow procedure will be executed at most

$$\sum_{i=0}^{\lfloor K/2 \rfloor} C(n_1, i) \;+\; \sum_{i=\lfloor K/2 \rfloor}^{0} C(n_0, i) \;\le\; 2 \cdot \sum_{i=0}^{\lfloor K/2 \rfloor} C(n, i) \;=\; O(2 \cdot n^{\lfloor K/2 \rfloor + 1})$$

times. The Max_Flow procedure has a complexity of $O(Km)$ because at most $(K+1)$ paths will be computed in order to know whether there is a K-feasible cut or not. Since the runtime of

Max_Flow procedure is the dominating factor in each iteration, the optimal min-cost K-feasible cut algorithm has a complexity of $O(2Kmn^{\lfloor K/2 \rfloor +1})$.

3.4.2. Further Speed-Up of the Min-Cost K-feasible Cut Algorithm

We present a theorem which can be used to further speed up the computation of the min-cost K-feasible cut. Suppose the algorithm fails to return a cut in $c=0$ pass (i.e., the amount of flow is larger than K). We complete the flow computation and obtain a flow distributtion. Let flow function f_0 denotes the flow in $c=0$ pass and $value(f_0)$ denotes the value of the max-flow. Then $value(f_0) > K$. Let $f_0(u)$ represent the amount of flow through node u. If there exists a K-feasible cut $(X_v,\overline{X}_v)$ of cost equal to one and $n(X_v,\overline{X}_v)=\{u_1,u_2,\ldots,u_l\}$, $l \le K$, let u_1 be the node of cost equal to one in the cut set. According to the flow conservation property,

$$\sum_{u \in n(X_v,\overline{X}_v)} f_0(u) = value(f_0).$$

Since each node in $(X_v,\overline{X}_v)$ except u_1 has flow capacity equal to one, we have

$$value(f_0) = f_0(u_1) + \sum_{i=2}^{l} f_0(u_i) = f_0(u_1)+(l-1) \le f_0(u_1)+(K-1).$$

That is, $f_0(u_1) \ge value(f_0)-K+1$. In general, we have the following theorem:

Theorem 2 Let $f_0(u)$ be the flow at node u at the end of $c=0$ pass in the optimal min-cost K-feasible cut algorithm. If there exists a K-feasible cut $(X_v,\overline{X}_v)$ of cost equal to c in N_v and $n(X_v,\overline{X}_v)=\{u_1,u_2,\ldots,u_l\}$, $l \le K$, let u_1, u_2, ... , u_c be those nodes of cost equal to one in the cut set. Then

$$f_0(u_1)+f_0(u_2)+\cdots+f_0(u_c) \ge value(f_0)-K+c.$$

Proof According to the flow conservation property, we have

$$value(f_0)=f_0(u_0)+f_0(u_1)+\cdots+f_0(u_c)+\sum_{i=c+1}^{l} f_0(u_i).$$

Because zero-cost node has a flow capacity of one, we have

$$value(f_0)=f_0(u_0)+f_0(u_1)+\cdots+f_0(u_c)+(l-c).$$

Since $l \le K$, we have

$$value(f_0) \le f_0(u_0)+f_0(u_1)+\cdots+f_0(u_c)+(K-c).$$

That is,

$$f_0(u_1)+f_0(u_2)+\cdots+f_0(u_c) \ge value(f_0)-K+c.$$

We may use this theorem to speed up the cut computation. For example, assume $K=5$ and $value(f_0)=7$ in the $c=0$ pass. Then in $c=1$ pass, we need not compute a cut in $N_v(u)$ for those nodes u of flow $f_0(u) < 3$. Similarly, in $c=2$ pass, those pair of nodes u_1, u_2 such that $f_0(u_1)+f_0(u_2) < 4$ are out of consideration. The flow function f_1 in $c=1$ pass can also be used in a similar way to reduce the search space in the followed passes.

We evaluate the impact of applying Theorem 2 in the optimal min-cost K-feasible cut algorithm by reporting number of times that max-flow routine is executed in $c=1$ and $c=2$ passes. Table 1 shows the numbers reported by CutMap for six small benchmarks. Numbers in row I and II are the numbers reported by CutMap without and with applying Theorem 2 for speed-up, respectively. For circuit 9*sym*, we are able to prune the search space by 99.7%. For large benchmarks, the impact is more significant.

3.4.3. Low-Cost K-feasible Cut Algorithm

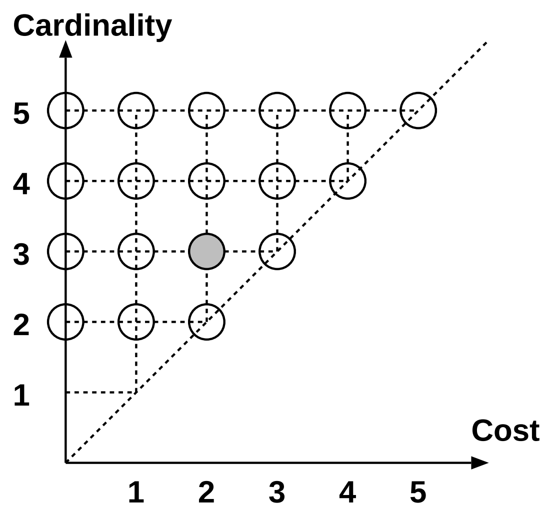
The min-cost K-feasible cut algorithm runs max-flow procedure for a polynomial number (in network size) of times. We develop a *low-cost* K-feasible cut algorithm which runs max-flow procedure for a number of times independent of the network size (in fact, it is polynomial in K, the LUT input capacity) but still returns a min-cost K-feasible cut in most cases. The idea is to improve the min-cost min-cut. In our experiments, we found that 95% of the K-feasible cuts returned by both methods have the same cost. Hence the low-cost K-feasible cut algorithm can serve as a fast heuristic for the optimal min-cost K-feasible cut algorithm.

In this algorithm, a cut is characterized by two parameters: cost (c) and cardinality (s). We identify each category of cuts by its location (c,s) on the cost-cardinality plane as shown in Figure 4(a). For example, the shaded circle located at $(2,3)$ represents the category of cuts which have cost $c=2$ and cardinality $s=3$. Figure 4(a) shows categories of K-feasible cuts for $K=5$. A category is not empty if there is a cut belongs to that category. In general, we are interested in a non-empty category located below or at the cardinality bound K and as left toward the vertical axis as possible.

The strategy of the low-cost K-feasible cut algorithm is to replace the min-cost min-cut by a cut of lower cost. It can be stated as (1) finding a min-cost min-cut followed by (2) trade-off between cardinality and cost. Let N_v be a network rooted at node v. Let V_0 and V_1 be the set of nodes which have a cost equal to zero and one, respectively. A min-cost min-cut exists since N_v is K-feasible. Suppose it belongs to a category located at (c_m, s_m). Then every category (c,s) with $s < s_m$ is empty. Every category (c,s) with $c < c_m$ and $s = s_m$ is also empty. Categories (c,s)

	z4ml	5xp1	misex1	vg2	rd84	9sym
I	730	891	811	2217	7430	24594
II	42	110	79	22	21	78

Table 1 Number of times that Max-Flow is executed in $c=1$ and $c=2$ passes in CutMap.



(a) (b)

Figure 4 (a) cost-cardinality plane (b) example that heuristics fails

with $c \geq c_m$ are not interesting to us. Only those categories (c,s) with $c \leq c_m$ and $s_m < s \leq K$ may contain K-feasible cuts of cost lower than the min-cost min-cut. We call them candidate categories. For example, assume $(c_m,s_m)=(2,3)$ in Figure 4(a). Then categories $(1,4),(1,5),(0,4),(0,5)$ are candidate categories. In general, there are $(c_m-1)\cdot(K-s_m)$ candidate categories.

In step (1), a min-cost min-cut is computed by assigning flow capacity to nodes in N_v as follows: $capacity(u)=1$ for $u \in V_0$ and $capacity(u)=1+\varepsilon$ for $u \in V_1$. The max-flow procedure will return a min-cut by this capacity assignment because ε is too small to influence the cut cardinality. But among all min-cuts, the min-cost min-cut has the smallest flow capacity. As a result the max-flow procedure returns a min-cost min-cut by the capacity assignment.

In step (2), we test whether a category is empty by assigning flow capacity properly. For example, assume $(c_m,s_m)=(2,3)$ in Figure 4(a). To test category $(0,4)$, we assign each node in V_0 a capacity of 2/3 and each node in V_1 a capacity of $1+\varepsilon$. This assignment causes cuts in category $(0,4)$ to have a flow capacity of 8/3 while it causes cuts in category $(2,3)$ to have a flow capacity of $8/3+2\varepsilon$. As a result the max-flow procedure will return a cut in category $(0,4)$ if it is not empty. In general, if the min-cost min-cut is in category (c_m,s_m) and category (c,s) is being tested, each node in V_0 will be assigned a capacity of $(c_m-c)/(c_m-c+s-s_m)$. This assignment causes cuts in category (c,s) to have a flow capacity $(c_m-c)\cdot\varepsilon$ less than the flow capacity of the min-cost min-cut.

In step (2), the algorithm first collects all candidate categories, sorts them in clockwise order relative to the min-cost min-cut category, and tests each category in ordered list for its emptiness. For example, the list will be $(0,4)$, $(1,4)$, $(0,5)$, $(1,5)$ in Figure 4(a). The first non-empty category found contains a cut better than the min-cost min-cut in terms of cost. However, this method does not guarantee a min-cost K-feasible cut in two conditions. First, when two categories are at same angular position (e.g. $(1,4)$ and $(0,5)$), the one with lower cost (which may not be K-feasible) will return. For example, let $K=5$ and $(c_m, s_m)=(2,4)$. Assume category $(0,5)$ is empty but category $(1,5)$ and $(0,6)$ are not. The emptiness test will return a cut in category $(0,6)$ while category $(1,5)$ does contain a 5-feasible cut of cost only one. Second, even when the emptiness test returns a K-feasible cut, it might not be the min-cost K-feasible cut. For example, let $K=5$ and $(c_m, s_m)=(2,2)$. Assume all candidate categories are empty except category $(1,3)$ and $(0,5)$. Then the emptiness test will return a cut in category $(1,3)$ since it is the first non-empty category in the sorted list. But the category $(0,5)$ contains the min-cost 5-feasible cut. Figure 4(b) illustrates an example of both effects. Shaded nodes are zero cost nodes. The min-cost min-cut is $(\{a\},\{b,c,d,e,f,g,h,i,j,k,m\})$ which belongs to category $(2,2)$. Category $(1,5)$ contains the min-cost 5-feasible cut $(\{a,b,c,e,f\},\{d,g,h,i,j,k,m\})$. The cut returned by the low-cost K-feasible cut algorithm will be $(\{a,b,c,d,e,f\},\{g,h,i,j,k,m\})$, which belongs to the non-empty category $(0,6)$ (not shown in Figure 4(a)), since its flow capacity is 6/4 while the min-cost 5-feasible cut has a flow capacity of $2+\varepsilon$ according to the node capacity assignment rule.

3.5. Summary of CutMap Algorithm

The CutMap algorithm produces a depth-optimal area-minimized mapping solution for a given K-bounded network N. CutMap has two phases. In the first phase, CutMap calculates a label for each node in N based on FlowMap algorithm. In the second phase, CutMap exploits depth criticality of mapped nodes and computes either a min-cost min-height K-feasible cut for nodes on critical path or a min-cost K-feasible cut for nodes not on critical path. The cut computation is based on an optimal min-cost K-feasible cut algorithm developed in this paper, which has a complexity of $O(2Kmn^{\lfloor K/2 \rfloor+1})$ where n is the number of nodes and m is the number of edges in the network. In addition, CutMap uses a predictive cost assignment method, which is based on the MFFC decomposition of the Boolean network, to achieve area minimization. To speed up the cut computation, we develop a theorem which can be used to reduce search space substantially, as well as a fast heuristic which returns a min-cost K-feasible cut in most cases. We conclude this section with the following theorem.

Theorem 3 For a given K-bounded Boolean network N and a depth bound D, the CutMap algorithm returns a functionally equivalent LUT network T in $O(2Kmn^{\lfloor K/2 \rfloor+2})$ time if $D \geq D_{\text{opt}}(N)$, where $D_{\text{opt}}(N)$ is the optimal depth derived by the FlowMap algorithm for N, n is the number of nodes in N and m is the number of edges in N.

Proof The CutMap algorithm is described in Figure 2. Let $n=|V(N)|$ and $m=|E(N)|$. In Phase 1, the computation of label for all nodes takes $O(Kmn)$ time. For each mapped node, it takes $O(2Kmn^{\lfloor K/2 \rfloor+1})$ time to execute the min-cost K-feasible cut algorithm (by Theorem 1).

There are at most n mapped nodes. For each mapped node, it might take $O(Kmn)$ time to update labels of its transitive successors. However, this time is dominated by the cut computation time. Other operation takes linear time. Totally it takes $O(2Kmn^{\lfloor K/2 \rfloor + 2})$ time in Phase 2, which dominates the time spent in Phase 1. As a result, the CutMap algorithm has a complexity of $O(2Kmn^{\lfloor K/2 \rfloor + 2})$.

4. Experimental Results

We have implemented the CutMap algorithm using the C language on Sun SPARC workstations. Given a general Boolean network, we uses the input/output and general utilities provided by MIS/SIS [BrRS87] to decompose it into a 2-input network of simple gates. Then we apply CutMap algorithm on the decomposed network to obtain a depth-optimal K-LUT mapping solution. Post-processing operations, including gate-decomposition, predecessor-packing [ChCD92] and FlowPack [CoDi94a], are applied to further reduce the number of LUTs in the solution. In our experiments, we target a 5-LUT based mapping solution to reflect, e.g., the X3000 FPGA family produced by Xilinx. It has been reported in [CoDi94a] that FlowMap outperforms other speed-oriented mapper such as Chortle-d, DAG-Map and MIS-pga-delay by as much as 50% in area and 7.1% in depth. Hence, we compare CutMap mapping solutions with only those by FlowMap.

We test the CutMap algorithm on 19 MCNC benchmarks, which were obtained by a sequence of technology independent optimization for both area and speed using MIS, and the resulting networks had been used by Chortle-d and FlowMap in their experiments. These benchmarks are already 2-input networks, so no preprocessing steps are required to guarantee a K-bounded network. Their sizes range from 48(z4ml) to 3263(des) nodes. Experimental data is shown in Table 2. We use FM, CM and PP to refer to FlowMap, CutMap and post-processing respectively. Results in columns FM and CM are the number of LUTs in the mapping solutions by FlowMap and CutMap without post-processing operations, respectively. Results in columns FM/PP and CM/PP are the number of LUTs in the FlowMap and CutMap solutions followed post-processing operations for area minimization. We also test FlowMap and CutMap targeting for Xilinx 3000 family where each CLB can be used as either a 5-input LUT or two 4-input LUTs with certain constraint on LUT inputs. The result is reported in columns FM/PP/GM and CM/PP/GM, respectively. In summary, CutMap uses 20% fewer LUTs than FlowMap without post-processing, 13% fewer LUTs with the post-processing routines, and 11% fewer CLBs when targeting for Xilinx 3000 FPGA family. In our experiment, both FlowMap and CutMap produces solutions with optimal depth D_{opt}.

We evaluate the performance of the low-cost K-feasible cut algorithm and present the results in Table 3. Columns OPT and HEU are the numbers of LUTs returned by CutMap using min-cost-K-feasible-cut and low-cut-K-feasible-cut algorithms in the cut computation respectively. We apply Theorem 2 in the min-cost-K-feasible cut algorithm. Both simulations

CKT	#PI	#PO	#node	FM	CM	FM PP	CM PP	FM PP GM	CM PP GM	D_{opt}
5xp1	7	10	104	29	24	25	24	20	18	3
9sym	9	1	200	75	68	61	65	50	50	5
9symml	9	1	191	66	62	58	59	51	50	5
C499	41	32	658	183	148	154	138	133	120	5
C880	60	26	548	258	214	233	205	193	171	8
alu2	10	6	393	173	146	162	145	129	123	8
alu4	14	8	726	292	259	267	252	220	210	10
apex4	9	19	1541	845	748	780	728	675	632	6
apex6	135	99	779	300	245	257	242	222	216	4
apex7	49	37	247	109	82	90	80	67	60	4
count	35	16	216	93	63	76	58	61	47	3
des	256	245	3263	1524	1059	1308	989	1060	835	5
duke2	22	29	392	219	191	187	178	149	142	4
e64	65	65	313	209	164	166	162	128	129	3
misex1	8	7	57	19	19	15	16	13	14	2
rd84	8	4	141	49	45	43	45	37	37	4
rot	135	107	647	315	256	268	237	205	179	6
vg2	25	8	120	54	40	45	38	34	29	4
z4ml	7	4	48	16	13	13	12	10	10	3
Total				4828	3846	4208	3673	3457	3072	
				1	0.80	1	0.87	1	0.89	

Table 2 Result from FlowMap and CutMap (LUTs)

exploit depth slack and consider roots of MFFC with 5 or more nodes as zero-cost nodes. Column OPT/PP and HEU/PP are numbers of LUTs after post-processing operations. Column OPT/(sec) and HEU/(sec) are the time in seconds used in computations. The min-cost-K-feasible-cut computation gives slightly better mapping solutions (in term of area) than the low-cost-K-feasible-cut heuristic, but using more than two times of the execution time. For large benchmark such as *apex*4, the time saving by the cut computation heuristic is significant. We evaluate the effectiveness of low-cost-K-feasible-cut algorithm to return a min-cost *K*-feasible cut as follows. Whenever CutMap calculates a min-cost *K*-feasible cut, it also calculates a low-cost *K*-feasible cut and compares their costs. The number of times that our heuristic returns minimum cost cuts is recorded and reported at the end of CutMap program. The result is shown in column OPT/HEU of Table 3. For 96% of the time the low-cost *K*-feasible cut heuristic computes a min-cost *K*-feasible cut. It supports the effectiveness of the low-cost *K*-feasible cut algorithm.

In Table 4, we showed the results of FlowMap_r [CoDi94b] and CutMap followed by depth relaxation routines in FlowMap_r on 11 MCNC benchmarks for relaxed depth $X = 0, 1, 2, 3$. It has been shown in [CoDi94b] that depth relaxation decreases the number of LUTs substantially on these benchmarks. FlowMap_r was originally designed to perform depth relaxation and remapping on a depth-optimal solution produced by FlowMap for area minimization. We

CKT	OPT	HEU	OPT PP	HEU PP	OPT (sec)	HEU (sec)	OPT/HEU
5xp1	27	27	24	24	1.8	1.3	38/38
9sym	68	70	61	61	7.1	5.1	68/66
9symml	63	63	58	58	5.3	4.9	64/64
C499	169	171	155	152	115.5	107.4	211/201
C880	221	223	202	203	105.3	63.2	315/309
alu2	152	156	145	148	53.7	33.3	203/197
alu4	267	279	247	251	111.5	92.6	397/381
apex4	770	787	729	730	793.1	135.9	814/768
apex6	258	260	249	249	39.2	20.1	441/419
apex7	87	91	83	83	6.7	6.3	135/126
count	78	81	74	73	4.8	3.7	91/85
des	1184	1192	1047	1046	212.4	131.9	1463/1429
duke2	197	200	180	182	20.1	15.4	225/215
e64	163	164	160	158	14.0	11.2	181/176
misex1	18	19	15	15	1.4	0.8	20/19
rd84	46	47	46	47	5.3	4.0	47/45
rot	263	270	257	256	88.8	43.5	460/437
vg2	48	48	43	40	5.0	4.2	63/61
z4ml	13	13	12	12	1.1	0.6	16/16
Total	4092	4161	3787	3788	1592.1	685.4	5252/5052

Table 3 Number of times the low-cost K-feasible cut algorithm returns a min-cost K-feasible cut in the mapping process

modified FlowMap_r so that it can be applied to any given mapping solutions, including those produced by CutMap for depth relaxation and area minimization. In column $D_{\mathrm{opt}}+X$ of Table 4, we record the number of LUTs in the LUT networks of depth $D_{\mathrm{opt}}+X$ produced by FlowMap_r and CutMap followed FlowMap_r (separated by a slash ‘‘/’’), respectively. If no solution is found for a particular depth, we leave a ‘‘-’’ in the entry. The total of each column is computed as the summation of numbers of LUTs in the mapping solutions which satisfy the depth bound and have minimal number of LUTs. Note that when $X = 0$, the mapping solution is still depth-optimal. It is not surprising that for $X = 0$ case, FlowMap_r reduces the number of LUTs in FlowMap’s mapping solution considerably, but CutMap followed FlowMap_r does not reduce the number of LUTs as effectively, since depth relaxation has been exploited by CutMap during the mapping process. The solutions produced by CutMap followed FlowMap_r are overall better than those produced by FlowMap_r for $X = 0, 1, 2$. As the relaxation bound X increases, the results become comparable because in this case FlowMap_r generates a new mapping solution after depth relaxation and remapping, which is quite different from the input solution.

5. Conclusion and Future Work

In this paper, we study the simultaneous depth and area minimization for LUT-based FPGA technology mapping problem. Our goal is to generate depth-optimal mapping solutions using as

CKT	D_{opt}+0	D_{opt}+1	D_{opt}+2	D_{opt}+3
rd84	47/43	39/41	37/38	-/-
count	77/63	60/60	-/-	-/-
apex7	83/79	79/76	-/-	-/-
duke2	188/182	167/173	151/153	-/-
alu2	154/146	143/142	137/136	135/133
C880	206/207	195/196	176/183	-/177
rot	248/238	229/230	225/218	216/217
C499	134/142	130/129	-/-	-/-
alu4	254/250	247/242	243/237	238/232
apex6	234/241	225/222	225/-	224/-
des	1140/989	1093/-	985/979	954/-
Total	2765/2580 1/0.93	2607/2500 1/0.96	2448/2431 1/0.99	2400/2416 1/1.01

Table 4 Number of LUTs in the solutions produced by FlowMap_r and CutMap+FlowMap_r.

few LUTs as possible for the given *K*-bounded Boolean networks. Our algorithm, called CutMap, not only achieve depth optimality as the FlowMap algorithm but also exploits depth relaxation and achieves area minimization in the mapping process, instead of considering them in the post-processing stages. The depth relaxation is based on updating of depth criticality dynamically for nodes in the network. The area minimization is achieved by using an optimal min-cost *K*-feasible cut algorithm developed in this work. The optimal min-cost *K*-feasible cut algorithm takes $O(2Kmn^{\lfloor K/2 \rfloor +1})$ time, where n is the number of nodes and m is the number of edges in the network. We can reduce the search space of the algorithm based on flow information collected in previous max-flow runs. A fast heuristic is also developed to calculate a low-cost *K*-feasible cut, which in 95% of times has the same cost as the min-cost *K*-feasible cut. For depth-optimal mapping solutions, CutMap uses 20% fewer K-LUTs than FlowMap without post-processing, and uses 13% fewer K-LUTs than FlowMap when post-processing operations for area minimization are applied to both solutions. When targeting for Xilinx X3000 FPGA family, CutMap uses 11% fewer CLBs than FlowMap.

The recent algorithm named FlowSYN [CoDi93b] uses functional decomposition based re-synthesis during the FlowMap mapping process and achieves good results in terms of both depth and area minimization. In most cases, FlowSYN outperforms CutMap in terms of both depth and area, but require longer runtime. We are in the process of extending ideas of FlowSYN and integrating resynthesis techniques during CutMap mapping process.

Acknowledgment

This work is partially supported by NSF Young Investigator Award MIP9357582, ARPA/CSTO under Contract DABT63-93-C-0055, and grants from AT&T, Fujitsu Laboratory at America, Xerox Foundation, and Xilinx under the California MICRO Program and NYI Award matching program. The authors would like to thank Eugene Ding for providing the FlowMap program and his valuable discussions.

References

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

[ChCD92] Chen, K. C., J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, ''DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization,'' *IEEE Design and Test of Computers*, pp. 7-20, Sep. 1992.

[CoDi93b] Cong, J. and Y. Ding, ''Beyond the Combinatorial Limit in Depth Minimization for LUT-Based FPGA Designs,'' *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 110-114, 1993.

[CoDi94a] Cong, J. and Y. Ding, ''An Optimal Technology Mapping Algorithm fo Delay Optimization in Lookup-Table Based FPGA Designs,'' *IEEE Trans. on Computer-Aided Design*, Vol. **13**, pp. 1-12, Jan. 1994.

[CoDi94b] Cong, J. and Y. Ding, ''On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping,'' *IEEE Trans. on VLSI Systems*, Vol. **2**, June 1994.

[CoLB94] Cong, J., Z. Li, and R. Bagrodia, ''Acyclic Multi-Way Partitioning of Boolean Networks,'' *Proc. ACM/IEEE 31st Design Automation Conf.*, pp. 670-675, June 1994.

[FaSa94] Farrahi, A. and M. Sarrafzadeh, ''Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping,'' *IEEE Trans. on CAD*, Vol. **13**(11) pp. 1319--1332, Nov. 1994.

[FrRC90] Francis, R. J., J. Rose, and K. Chung, ''Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays,'' *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 613-619, June 1990.

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

[Ka91a] Karplus, K., ''Xmap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays,'' *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 240-243, June 1991.

[LaPV93] Lai, Y.-T., M. Pedram, and S. Vrudhula, ''BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis,'' *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 642-647, June 1993.

[MuNS90] Murgai, R., Y. Nishizaki, N. Shenay, R. Brayton, and A. Sangiovanni-Vincentelli, ''Logic Synthesis Algorithms for Programmable Gate Arrays,'' *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 620-625, 1990.

[MuSB91] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, ''Performance Directed Synthesis for Table Look Up Programmable Gate Arrays,'' *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 572-575, Nov. 1991.

[ScKC92] Schlag, M., J. Kong, and P. K. Chan, ''Routability-Driven Technology Mapping for Lookup Table-Based FPGAs,'' *Proc. 1992 IEEE International Conference on Computer Design*, pp. 86-90, Oct. 1992.

[Xi93b] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, San Jose, CA (1993).