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

Abstract

In this paper, we present an improvement of the FlwoMap algorithm, named CutMap, which combines depth and area minimization during the mapping process by computing min-cost min-height K-feasible cuts for critical nodes for depth minimization and computing min-cost K-feasible cuts for non-critical nodes for area minimization. CutMap guarantees depth-optimal mapping solutions in polynomial time as the FlowMap algorithm but uses considerably fewer K-LUTs. We have implemented CutMap and tested it on the MCNC logic synthesis benchmarks. For depth-optimal mapping solutions, CutMap uses 15% fewer K-LUTs 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

Previous work on lookup table (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], Xmapa [Ka91a], FGMap [LaPV93] focus on minimizing the number of LUTs. Mappers such as Chortle-d [FrRV91b], Mis-pga-delay [MuSB91a], 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 K-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, such as 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 during the mapping process by computing min-cost min-height K-feasible cuts for critical nodes for depth minimization and computing min-cost K-feasible cuts for non-critical nodes for area minimization. The min-cost K-feasible cut computation encourages sharing for K-LUTs when mapping different parts of the circuits. CutMap still guarantees depth-optimal mapping solutions in polynomial time but uses considerably fewer K-LUTs. We have implemented CutMap and tested it on the MCNC logic synthesis benchmarks. For depth-optimal mapping solutions, CutMap uses 15% fewer K-LUTs 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. The proofs of all theorems in this paper are omitted due to page limitation and can be found in [CoHw95].

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-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-hard[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 also 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 $(X, \bar{X})$ is a partition of $V(N)$ such that $s \in X$ and $t \in \bar{X}$. For a given cut $(X, \bar{X})$, the edge set of the cut, denoted $es(X, \bar{X})$, is defined as $es(X, \bar{X}) = \{(u,v) \mid (u,v) \in E(N), u \in X, v \in \bar{X}\}$ and the node set of the cut, denoted $ns(X, \bar{X})$, is defined as $ns(X, \bar{X}) = \{u \mid (u,v) \in es(X, \bar{X}), v \in V(N)\}$. The *edge(node) cut-size* of a cut is the cardinality of its edge(node) set. For a given network with a source and a sink, a *min-edge(node) cut* is a cut of minimum edge(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, we assign infinite capacity to every edge, divide each node into two nodes which are linked by a directed edge with 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.

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 $(X, \bar{X})$ is K-feasible if its cut-size is no more 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 necessary a min-cost cut. 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 and the min-cost K-feasible cut algorithm, which is used by the CutMap algorithm to map each implemented node during the mapping process. 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 $D_{opt}(N)$ denotes the largest $label(v)$ for all $v \in V(N)$, i.e., $D_{opt}(N) = \max\{label(u) \mid u \in V(N)\}$. 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. $sp(v) = ns(X_v, \bar{X}_v)$. 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 and FlowPack) 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 on non-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 a cut for each node in the first phase, instead, we compute either a min-height K-feasible cut or a min-cost K-feasible cut depending on the depth criticality of the mapped node. 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 $D_{opt}(N)$ is then computed. If $D_{opt}(N) > D$, the given depth bound, CutMap stops and claims that no solution exists with respect to the given depth bound. Otherwise, CutMap proceeds to assign $latest(v) = D_{opt}(N)$ for every $v \in V(N)$. In the second phase, CutMap initializes Q by the set of all POs and repeats the following four steps until Q contains only PIs. (1) Remove the node v of minimal slack from Q. (2) Assign costs of nodes in N_v based on predicting which nodes will be implemented (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 with $label(u) = label(v)$ in N_v as in FlowMap 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.

Whenever a cut is computed for a mapped node v, 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 LUT. 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 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 $D \ge D_{opt}(N)$ 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 min-cost K-feasible cut procedure (discussed in Section 3.4). We have the following result:

Theorem 1 For a given K-bounded Boolean network N and a depth bound D, the CutMap algorithm returns a functionally equivalent LUT network T in $O(lM)$ time if $D \ge D_{opt}(N)$, where $D_{opt}(N)$ is the optimal depth of N, l is the number of LUTs in the mapping solution, and M is the time taken by the min-cost K-feasible cut computation.

3.3. Predictive Cost Assignment

A min-cost K-feasible cut is computed for every node $v \in Q$ in the execution of CutMap algorithm. During the execution of CutMap algorithm, a node is assigned a cost of zero if it has been implemented or likely to be implemented later on by a LUT. 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 using LUTs. 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 $u \ne v \in FFC_v$, $output(u) \subseteq FFC_v$. 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 $X_v \cap MFFC_i = \varnothing$ 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), $mn^{K+1}=3\cdot10^{21}$ while $2Kmn^{\lfloor K/2\rfloor+1}=3\cdot10^{13}$. 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 $ns(X_v,X_v)\cap V_1=\varnothing$. 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 $|ns(X_v,\bar{X}_v)\cap V_1|=1$. 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* with zero-cost to form a new network $N_v(S)$ and test to see if $N_v(S)$ has a zero-cost *K*-feasible cut. 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 nodes 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 $|ns(X_v,\bar{X}_v)\cap V_0|\le K-c$. 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 1 gives the pseudo code for the optimal min-cost *K*-feasible cut algorithm. We have the following result:

Theorem 2 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.

3.4.2. 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 in $c=0$ pass to return a cut. 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 $(X_v,\bar{X}_v)$ of cost one, let u_1 be the node of cost one in such a cut. According to the flow conservation property,

$$\sum_{u\in ns(X_v,\bar{X}_v)} f_0(u)=value(f_0)$$

Since other nodes in $(X_v,\bar{X}_v)$ have capacity one, we have $f_0(u_1)+(K-1)\ge value(f_0)$, i.e., $f_0(u_1)\ge value(f_0)-K+1$. In general, we have the following theorem:

Procedure Min_Cost_K_Feasible_Cut(N_v, v, K)

$V_0 = \{ u \mid cost(u) = 0 \}$, $V_1 = \{ u \mid cost(u) = 1 \}$;

for $c = 0$ **to** $\lfloor K/2 \rfloor$ **do**
  for each subset S of V_1 such that $|S| = c$ **do**
    $V_1 = V_1 - S$, $V_0 = V_0 \cup S$;
    $capacity(u) = 1$ for all $u \in V_0$;
    $capacity(u) = \infty$ for all $u \in V_1$;
    $f_c(S)$ = Max_Flow(N_v);
    if $value(f_c(S)) \le K$
      a cut of cost c is found, **return** the cut;
    $V_1 = V_1 \cup S$, $V_0 = V_0 - S$;
  end for
for $c = \lfloor K/2 \rfloor + 1$ **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 $u \in S$;
    $capacity(u) = \infty$ for all $u \in V_0$;
    $capacity(u) = 1$ for all $u \in V_1$;
    $f_c(S)$ = Max_Flow(N_v);
    if $value(f_c(S)) \le c$
      a cut of cost c is found, **return** the cut;
    $V_0 = V_0 \cup S$;
  end for
for $c = K$ **to** K **do**
  return a min-cut;

Figure 1 Optimal Min Cost K-feasible Cut Algorithm

Theorem 3 Let $f_0(u)$ be the flow at node u at the end of $c = 0$ pass. If there exists a K-feasible cut of cost c, and u_1, u_2, ... , u_c are those unit-cost nodes in the cut set, then $f_0(u_1) + f_0(u_2) + \cdots + f_0(u_c) \ge value(f_0) - K + c$.

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 $c = 1$ can also be used in a similar way to reduce the search space in the followed passes. In our experiments, we are able to reduce the search space by as much as 350 times even for small benchmarks. For large benchmarks, the impact is more significant.

3.5. Summary of CutMap Algorithm

The CutMap algorithm produces a depth-optimal mapping solution for a given K-bounded network N and a given depth bound D. CutMap has two phases. In the first phase, CutMap calculates a label for each node in N using FlowMap algorithm. In the second phase, CutMap exploits depth criticality of mapped nodes and computes either a 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. The complexity of such algorithm is $O(2Kmn^{\lfloor K/2 \rfloor + 1})$ where n is the number of nodes and m is the number of edges in the network. We give a theorem based on which we can speedup the computation for min-cost K-feasible cut. CutMap uses a a predictive cost assignment method, which is based on the MFFC decomposition of the Boolean network, to assign cost of nodes and achieve area minimization.

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 18 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 1. 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. CutMap uses 22% fewer LUTs than FlowMap does without post-processing and 15% fewer LUTs with the post-processing routines. In this experiment, both FlowMap and CutMap produces solutions with optimal depth D_{opt}.

We evaluate the impact of applying Theorem 3 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 2 shows the numbers reported by

CKT	FM	CM	FM PP	CM PP	D_{opt}
5xp1	29	24	25	24	3
9sym	75	68	61	65	5
9symml	66	62	58	59	5
C499	183	148	154	138	5
C880	258	213	233	205	8
alu2	173	148	162	145	8
alu4	292	255	267	252	10
apex6	300	244	257	242	4
apex7	109	82	90	80	4
count	93	62	76	58	3
des	1524	1059	1308	989	5
duke2	219	191	187	178	4
e64	209	164	166	162	3
misex1	19	19	15	16	2
rd84	49	45	43	45	4
rot	315	255	268	237	6
vg2	54	40	45	38	4
z4ml	16	13	13	12	3
Total	3983	3092	3428	2945	
	1	0.78	1	0.86	

Table 1 Result from FlowMap and CutMap (LUTs)

CutMap for six small benchmarks. Numbers in row I and II are the numbers reported by CutMap without and with applying Theorem 3 for speed-up, respectively. It is clear that CutMap gains significant speedup based on the result of Theorem 3.

In Table 3, 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 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 $D_{opt}+X$ of Table 3, we record the number of LUTs in the LUT-networks of depth $D_{opt}+X$ produced by FlowMap_r and CutMap followed by 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.

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

Table 2 Number of Times that Max-Flow is Executed in $c = 1$ and $c = 2$ passes in CutMap

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 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 dynamic updating of depth criticality for nodes in the network. The area minimization is achieved by using an optimal min-cost K-feasible cut algorithm designed in this work. Our 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

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	2607/2500	2448/2431	2400/2416
	1/0.93	1/0.96	1/0.99	1/1.01

Table 3 Number of LUTs in the solutions produced by FlowMap_r and CutMap+FlowMap_r.

network. We further reduce its runtime using the max-flow information.

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 suggestions and 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 for 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.

[CoHw95] Cong, J. and Y.-Y. Hwang, ‘‘Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping,’’ *Proc. ACM 3rd Int’l Symp. on Field Programmable Gate Arrays*, pp. 68-74, Feb. 1995.

[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.

[MuSB91a] 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.