An Efficient Cut Enumeration for Depth-Optimum Technology Mapping for LUT-based FPGAs

Takata, Taiga
Graduate School of Information Science and Electrical Engineering Kyushu University

Matsunaga, Yusuke
Faculty of Information Science and Electrical Engineering Kyushu University

出版情報：ACM Great Lakes Symposium on VLSI. 2009, pp.351-356, 2009-05-12
バージョン：
権利関係：

An Efficient Cut Enumeration for Depth-Optimum Technology Mapping for LUT-based FPGAs

Taiga Takata
Graduate School of Information Science
and Electrical Engineering
Kyushu University
Fukuoka, Japan
taiga@c.csce.kyushu-u.ac.jp

Yusuke Matsunaga
Faculty of Information Science
and Electrical Engineering
Kyushu University
Fukuoka, Japan
matsunaga@c.csce.kyushu-u.ac.jp

ABSTRACT

Recent technology mappers for LUT based FPGAs employ cut enumeration. Although many cuts are often needed to find good network, enumerating all cuts with large size consumes run-time very much. The number of cuts exponentially increases with the size of cuts, which causes long run-time. Furthermore, an inefficiency of bottom-up merging in existing algorithms makes the run-time much longer. This paper presents a novel cut enumeration. The proposed algorithm is efficient because it enumerates cuts without bottom-up merging. Our algorithm has two modes; exhaustive enumeration and partial enumeration. Exhaustive enumeration enumerates all cuts. Partial enumeration enumerates partial cuts with a guarantee that a depth-minimum network can be constructed. The experimental results show that exhaustive enumeration runs about 3 times and 8 times faster than existing bottom-up algorithm [1] [2] for $K = 8, 9$, respectively. The quality of network are the same. Furthermore, partial enumeration runs about 6 times and 18 times faster than bottom-up algorithm for $K = 8, 9$, respectively. Area of network derived by the set of cuts enumerated by partial enumeration is only 4 % larger than that derived by exhaustive enumeration on average, and the depth is the same.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Automatic Synthesis*

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

Recently, designs using LUT (LookUp-Table) based FPGAs (Field Programmable Gate Array) are becoming popular. LUT-based FPGAs require common photo-masks for multiple designs, while ASICs (Application Specific Integrated Circuits) require specific photo-masks for individual design. Once an LUT-based FPGA is reconfigured for a design, it is available to use immediately. For these reasons, LUT-based FPGAs are often used for prototypes or manufactures required to be developed faster. Main drawbacks of LUT-based FPGAs are performance and power consumption. Thus, technology mapping for LUT-based FPGAs is required to generate high-quality solution in short run-time. Technology mapping for LUT-based FPGAs is a process to convert a given Boolean network into a functionally equivalent network comprised of K-input LUTs [1]. The first-priority for delay-aware technology mappers [3] [4] [5] [1] [2] [6] [7] [8] [9] is depth-minimization, and the second-priority is area-minimization. Depth of an LUT network means the length of the longest path. Area means the number of LUTs.

A K-feasible cut is a cut whose size is up to K. A cone induced by a K-feasible cut is called K-feasible cone. Technology mapping is often treated as a problem to cover given Boolean network with K-feasible cones. A K-feasible cut is simply denoted by a cut in the rest of this section. Recent methods enumerate cuts at first, and then cover given Boolean network with K-feasible cones induced by the enumerated cuts. Because it is difficult to know which cut is good before covering, multiple cuts for each node are often needed to be held. Enumerating all cuts with large size, i.e. 8 or more, consumes run-time very much.

The one of the factors in long run-time of existing methods is the inefficiency of algorithms. Existing methods to enumerate all cuts [10] [1] [2] [6] [8] are based on bottom-up algorithm. Bottom-up algorithm generates a cut of a node by merging cuts at its fanins. To generate all cuts for each node, bottom-up algorithm calculates Cartesian products of the sets of the cuts of its fanins. The number of cuts are much fewer than the size of the Cartesian products in most cases. It is a drawback for bottom-up algorithm to need procedures proportional to the size of Cartesian products. Unlike bottom-up algorithms, top-down algorithm does not need to compute Cartesian products of cuts. Top-down algorithm is based on expansions of cuts. It generates new cuts by iterative expansions of some part of a cut to its fanin. Top-down algorithm looks superficially efficiency, but that has a problem. The problem is that the condition to stop expansions of cuts is not known. There are some cases where a cut whose size is K or under is obtained by the expansion

[1] In this paper, a K-input LUT is denoted by an LUT, and a network comprised of K-input LUTs is denoted by an LUT network.

of a not-K-feasible cut. Thus, top-down algorithm must execute expansions until those reach primary inputs without proper condition to stop expansions. This is inefficient for the increases of circuit sizes. For above reason, top-down algorithm have been rarely-used to enumerate all cuts.

The other factor in long run-time is large number of cuts. The number of cuts exponentially increases with K. Recent algorithm [9] avoids to enumerate all of cuts and compute very limited cuts; $8 \sim 4095$ cuts per node. Because the algorithm [9] enumerates cuts with heuristic techniques, it cannot guarantee to generate depth-optimum LUT network, and area is likely to be larger than that generated by recent methods with using all of cuts [2]. For applications needed to generate quality solutions, the cut enumerations in the paper [9] seems to be inadequate.

This paper presents a top-down algorithm which narrows down the search range compactly by setting the condition to stop expansions of cuts appropriately. It is more efficient than bottom-up algorithm because it enumerates cuts without calculating Cartesian products of cuts. The proposed algorithm has two modes; exhaustive enumeration and partial enumeration. The exhaustive enumeration enumerates all of cuts. The partial enumeration enumerates limited cuts while it guarantees that any depth-minimum covering algorithm with using enumerated cuts generates a depth-optimum LUT network. The partial enumeration enumerates a lot of *label-cuts* for each node. *label-cuts* are cuts needed to implement each node to be depth-optimum. It also enumerates other cuts which are valid to minimize area. The experimental results show that the exhaustive enumeration enumerates all cuts about 3 times and 8 times faster than bottom-up algorithm (used in methods [1] [2]) for $K = 8, 9$, respectively. If covering algorithms are same, the quality of LUT network is same because the sets of cuts enumerated by them are same. The experimental results also show that partial enumeration runs about 6 times and 18 times faster than bottom-up algorithm for $K = 8, 9$, respectively. The depth is same because the depth of LUT network derived by either of them is optimum. The area of LUT network derived by partial enumeration is only 4% larger than that derived by bottom-up algorithm.

The rest of this paper is organized as following. Section 2 introduces basic definitions. Section 3 reviews bottom-up algorithm to enumerate all cuts. Section 4 presents our algorithm. Section 5 shows experimental results. Section 5 concludes this paper.

2. PRELIMINARIES

The inputs of technology mapping are a Boolean network and a natural number K. A given Boolean network is called a subject graph. The natural number K corresponds to the maximum number of inputs of an LUT. The output of technology mapping is an LUT network.

A subject graph is the DAG (Directed Acyclic Graph). A subject graph is comprised of the set of nodes $v \in V$ and the set of directed edges $(i, j) \in E$. A node of a subject graph represents a primitive Boolean function and has up to K inputs. If a node $i \in V$ is an input of a node $j \in V$, there is an edge $(i, j) \in E$. The fanin of a node v is the set of immediate predecessors of v. The set of fanins of v is defined by $FI(v) = \{u \mid \exists(u, v) \in E\}$. There is generally only constraint for the size of the fanin of each node in a subject graph to be up to K. For understandability in this paper, it is assumed that the size of the fanin of each node in a subject graph is up to 2. The following discussion can be easily expanded to some general case without losing generality. The fanout of a node v is the set of immediate successors of v. The set of fanouts of v is defined by $FO(v) = \{w \mid \exists(v, w) \in E\}$. A node v whose $FI(v) = \phi$ is called a primary input. A node v whose $FO(v) = \phi$ is called a primary output. PI and PO denotes the set of primary inputs and the set of primary outputs, respectively. The transitive fanin of a node v is the set of all predecessors of v. More exactly, a transitive fanin of v, denoted by $TFI(v)$, is defined by $\{v\} \cup \bigcup_{u \in FI(v)} TFI(u)$. A transitive fanin graph of $v \in V$ is the subgraph induced by $TFI(v)$, denoted by $TFIG(v)$.

A separator s for $TFIG(v)$ is a set of nodes where any path from any primary input to v includes one or more nodes in s. A minimal separator s for $TFIG(v)$ is a separator for $TFIG(v)$ which does not contain any other separator for $TFIG(v)$. A **cut** (s, v) is a pair of node v and a set of nodes s where s is a minimal separator for $TFIG(v)$. For a node v, the cut $(v, \{v\})$ is called the *trivial* cut of v. For a cut (v, s), v and s are called the root and the leaf of the cut, denoted by $RT((v, s))$ and $LEAF((v, s))$, respectively. For a cut c, $|LEAF(c)|$ is called the *cut-size* of c. A **K-feasible cut** c is a cut where $|LEAF(c)|$ is up to K. A K-feasible cut is simply called as a cut in the rest of this section. $\Phi_K(v)$ denotes the set of all of cuts of v. For a set of cuts C and a cut $c \in C$, the **cut fanin** $CFI(c, C)$ is defined by $\{c' | c' \in C, RT(c') \in LEAF(c)\}$. For a feasible cut c, the **feasible cone** $KFC(c)$ is the subgraph induced by the nodes between $RT(c)$ and $LEAF(c)$. A K-feasible cone is exactly defined as the subgraph induced by $V_{interv}(RT(c), LEAF(c))$, where $V_{interv}(v, V)$ is derived by $V_{interv}(v, V) = \{v\} \cup \bigcup_{u \in FI(v)-V} V_{interv}(u, V)$. For the K-feasible cone FC derived by a cut c, the root of c is also called the root of FC, and denoted by $RT(FC)$. In above case, the leaf $LEAF(c)$ is called the inputs of K-feasible cone FC, and denoted by $INPUT(FC)$.

For a K-feasible cone FC, $INPUT(FC)$ is up to K. Thus, a K-inputs LUT can implement the Boolean function of any K-feasible cone. If a K-inputs LUT L implement the Boolean function of a K-feasible cone $KFC(c)$, the output signal of L corresponds to $RT(KFC(c))$, i.e. $RT(c)$, and the input signals of L correspond to $INPUT(KFC(c))$, i.e. $LEAF(c)$. If a set S of cuts meets below three conditions, S is called as the **realizable set**.

- $\forall i \in PO, (\exists c \in S, i = RT(c)) \vee i \in PI$
- $\forall c \in S,\ \forall i \in LEAF(c), (\exists c' \in S, i = RT(c')) \vee i \in PI$
- There is no *trivial* cut in S.

An LUT network can be generated from a realizable set

[2]For circuits in MCNC benchmarks and ITC'99 benchmarks, experimental results show that area of LUT network generated by the algorithm [9] is 11.4% larger than that generated by a depth-minimum mapper [7].

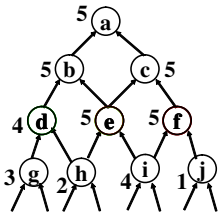

Figure 1: An example of a subject graph

S. The technology mapping problem can be defined as the problem to find a realizable set of cuts. The level of a node L in an LUT network is the length of the longest path from any node in PI to L. The level of a node in PI is zero. The depth of an LUT network is the longest path from any node in PI to any node in PO. The depth of an LUT network is equal to the largest level in the LUT network.

For each node v in a subject graph, we define the label of v, denoted by $L_K(v)$, to be the depth of the depth-optimum LUT network for $TFIG(v)$. The label of each node in PI is zero. The level for the K-feasible cone whose root is v in the depth-optimum LUT network is at least $L_K(v)$. The maximum label of all PO in subject graph is the depth of the depth-optimum LUT network. The height of a cut c, denoted by $H(c)$, is defined to be the maximum label in $LEAF(c)$. The height of a cut c is defined by $\max_{i \in LEAF(c)} L_K(i)$. Since the leaf of a cut c whose root is v corresponds to the set of inputs of LUT at v, the label of a node v can be computed by $\min_{c \in \Phi_K(v)} H(c) + 1$. A cut c of node v is called *label-cut* if $H(c) + 1$ is equal to $L_K(v)$. If $TFIG(v)$ is covered in depth-optimum way, the node v must be implemented by $KFC(c)$ where c is one of the *label-cuts* of v. Figure 1 shows the example of a subject graph. The circles represent the nodes and the arrows represent the edges in a subject graph. The numbers beside nodes represent labels of each nodes in figure 1. Because $L_K(d), L_K(e), L_K(f)$ are $4, 5, 5$ respectively, the height of 3-feasible cut $(a, \{d, e, f\})$ is 5. There are no other 3-feasible cuts of a whose height is 4 or under, the label of a is 5, and $(a, \{d, e, f\})$ is one of the *label-cuts* of a.

3. REVIEW OF BOTTOM-UP ALGORITHM FOR CUT ENUMERATION

For a K-feasible cut C, let $ISKF(C)$ be a Boolean function which returns true if and only if C is a K-feasible cut. For a node v and the two sets of K-feasible cuts A, B, an operation of *cut-merging* $A \bowtie_K B$ is defined by the following equation [1].

$$A \bowtie_{(K,v)} B = \{(v, LEAF(s) \cup LEAF(t)) \mid s \in A, t \in B, ISKF((v, LEAF(s) \cup LEAF(t))) = true\}$$

Then, All K-feasible cuts for each node can be enumerated in topological order from PI to PO according to the following theorem.

Theorem 1. The set of K-feasible cuts of a node v, $\Phi_K(v)$ can be computed by the following equation [1], where $FI(v) = \{u_1, u_2\}$.

$$\Phi_K(v) = \begin{cases} \{(v, \{v\})\} & v \in PI \\ \{(v, \{v\})\} \cup (\Phi_K(u_1) \bowtie_{(K,v)} \Phi_K(u_2)) & other. \end{cases}$$

There is a problem in the operations of *cut-merging* $\Phi_K(u_1) \bowtie_{(K,v)} \Phi_K(u_2)$. Let us set $M = |\Phi_K(u_1)|$ and $N = |\Phi_K(u_2)|$. The computing complexity of the operations of *cut-merging* to generate $\Phi_K(v)$ is $O(M \times N)$. If the size of $\Phi_K(v)$ is $O(M \times N)$, the computing complexity $O(M \times N)$ for computing $\Phi_K(v)$ is valid. But in most cases, the size of $\Phi_K(v)$ is much smaller than $O(M \times N)$. For example, $\{(v, LEAF(s) \cup LEAF(t)) \mid s \in u_1, t \in u_2\}$ has possibilities to contain cuts whose size exceeds K. Figure 1 shows the example of a subject graph. If we enumerate $\Phi_3(a)$ according to theorem 1, we must calculate $\Phi_3(b) \bowtie_{(3,a)} \Phi_3(c)$. There are the sets of 3-feasible cuts $\Phi_3(b) = \{(b, \{b\}), (b, \{d, e\}), (b, \{d, h, i\}), (b, \{g, h, e\}), (b, \{g, h, i\})\}$, $\Phi_3(c) = \{(c, \{c\}), (c, \{e, f\}), (c, \{h, i, f\}), (c, \{e, i, j\}), (c, \{h, i, j\})\}$. The 25 sets of cuts are generated by $\Phi_3(b) \times \Phi_3(c)$ because the sizes of $\Phi_3(b)$ and $\Phi_3(c)$ are both 5. But the set of 3-feasible cuts $\Phi_3(a) = \{(a, \{a\}), (a, \{b, c\}), (a, \{b, e, f\}), (a, \{c, d, e\}), (A, \{d, e, f\})\}$, and the unnecessary 21 sets of nodes are checked either each of them constructs a K-feasible cut. This inefficiency is unavoidable as far as enumerating K-feasible cuts by bottom-up algorithms.

4. AN EFFICIENT TOP-DOWN ALGORITHM FOR CUT ENUMERATION

4.1 Expansions of Cuts

An expansion of a cut C and a node $u \in LEAF(C)$, denoted by $EXP(u, C)$, is the operation to generate a new cut of nodes C' from C by removing a node u from $LEAF(C)$ and adding $FI(u)$ to $LEAF(C)$. $EXP(u, C)$ is calculated by $EXP(u, C) = (RT(C), (LEAF(C) - \{u\}) \cup FI(u))$.

Any cut of a node v but $(v, \{v\})$ can be generated by some expansion of cut. Hence, the set of all the cuts of a node v can be generated by starting at $(v, \{v\})$ and executing iterating expansions toward PI. Figure 2 shows the set of cuts generated by the iterative expansions for the subject graph shown in figure 1. The edges represents that the cut at the head is generated by the expansion of the cut at the tail. The alphabet by an edge means the name of the node removed on the expansion. If expansions of cuts are executed in a naive order, there is a possibility that same cuts are generated redundantly. For example, if $C = EXP(b, (a, \{b, c\}))$ is generated at first, and $C'' = EXP(c, C)$ is generated at second, then $C'' = (a, \{d, e, f\})$ is obtained. On the other hand, if $C' = EXP(c, (a, \{b, c\}))$ is generated at first, and $EXP(b, C')$ is generated at second, then C" is obtained too. To avoid the redundant enumeration, the order of priority to apply expansions are determined for all nodes. The order is arbitrary as far as the order relations do not change during cut enumeration to avoid redundant enumeration.

If there are any re-convergent points in subject graph, a K-feasible cut can be generated by some cut (v, s) where $|s| \geq K$. Executing iterative expansions of cuts until the expansions reach PI is inefficient. Thus, the condition to stop iterative expansions is needed to enumerate K-feasible cuts efficiently.

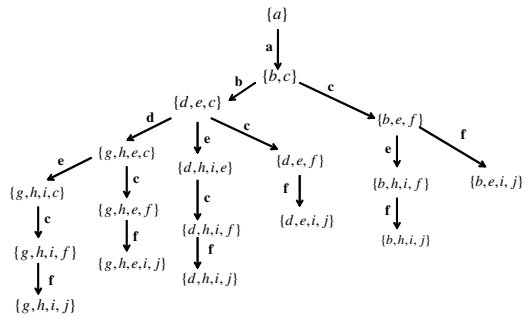

Figure 2: Cut enumeration based on expansions of cuts

4.2 Exhaustive Cut Enumeration

An efficient top-down algorithm for enumerating all K-feasible cuts is shown in this section. It is based on expansions of cuts. The top-down algorithm for enumerating K-feasible cuts of v starts with the K-feasible cut $(v, \{v\})$, and generate new K-feasible cuts of v by iterative expansions of the cuts. The top-down algorithm narrows down the search range compactly by setting the condition to stop expansions of cuts appropriately.

Let $\Psi(X) = \bigcup_{i=1}^{n} LEAF(C_i)$, where $X = \{C_1, C_2, ..., C_i\}$ be a set of K-feasible cuts of nodes. Then, $\Psi(\Phi_K(v))$ means the set of nodes contained in leaves of all of the K-feasible cuts of v. The following lemma is derived by theorem 1.

Lemma 1. The following condition is met for any node $v \notin PI$.

$$\begin{aligned} \Psi(\Phi_K(v)) &\subseteq \{v\} \cup \Psi(\Phi_K(u_1)) \cup \Psi(\Phi_K(u_2)) \quad (1) \\ where & \quad FI(v) = \{u_1, u_2\} \end{aligned}$$

proof. The following equation (2) is derived by theorem 1 for $v \notin PI$.

$$\begin{aligned} \Psi(\Phi_K(v)) &= \Psi(\{(v, \{v\})\} \cup (\Phi_K(u_1) \bowtie_{(K,v)} \Phi_K(u_2))) \\ &= \{v\} \cup \Psi(\Phi_K(u_1) \bowtie_{(K,v)} \Phi_K(u_2)) \quad (2) \end{aligned}$$

On the other hand, the following condition is met, where A and B are sets of K-feasible cuts.

$$\begin{aligned} \Psi(A \bowtie_{(K,v)} B) &\subseteq \Psi(\bigcup_{s \in A} \bigcup_{t \in B} LEAF(s) \cup LEAF(t)) \\ &= \Psi(\bigcup_{s \in A} LEAF(s) \cup \bigcup_{t \in B} LEAF(t)) \\ &= \Psi(A) \cup \Psi(B) \quad (3) \end{aligned}$$

The following equation is derived by the equations (2) and (3).

$$\Psi(\Phi_K(v)) \subseteq \{v\} \cup \Psi(\Phi_K(u_1)) \cup \Psi(\Phi_K(u_2)) \quad (4)$$

Theorem 2 is derived by lemma 1.

```
Partial_Enumeration ( SubjectGraph G, int K ){
  (v_1, v_2, ..., v_n) ← topological_sort (G) ;
  for each v_i {
    // Calculate the label of v_i with labeling in FlowMap[3]);
    CalcLabel(v_i);
  }
  for each v_i { //cuts enumeration phase
    u_1, u_2 ← FI(v_i) ;
    Check-mark all the nodes w,
        where w ∈ {v} ∪ Ψ(Ω_K(u_1)) ∪ Ψ(Ω_K(u_2));
    LabelCutsEnumeration(v_i);
    AreaCutsEnumeration(v_i);
  }
}
```

Figure 3: The framework for partial enumeration

Theorem 2. There is no K-feasible cut of v generated with any expansion of cuts $EXP(i, C)$ for $i \notin \{v\} \cup \Psi(\Phi_K(u_1)) \cup \Psi(\Phi_K(u_2))$.

According to theorem 2, all K-feasible cuts for each node are enumerated efficiently in topological order from PI to PO. The computing complexity of $\{v\} \cup \Psi(\Phi_K(u_1)) \cup \Psi(\Phi\{K\}(u_2))$ is $O(m+n)$, where $m = |\Psi(\Phi_K(u_1))|$, $n = |\Psi(\Phi_K(u_2))|$. If the number of K-feasible cuts generated by expansions of cuts is assumed to be E, the computing complexity of the top-down algorithm is $O(E)$. Not all the cuts generated by expansions of cuts are K-feasible cuts, but the computing complexity of the top-down algorithm is possible to be lower than that of the bottom-up algorithm, because it does not need to compute *cut-merging*, and the condition to stop the iterative expansions of cuts is tight.

4.3 Partial Cut Enumeration

An efficient algorithm is presented for enumerating limited K-feasible cuts while it guarantees a depth-minimum LUT network can be constructed. Because only K-feasible cuts are discussed, K-feasible cuts are simply denoted by cuts in the rest of this paper.

4.3.1 Framework

Our algorithm enumerates *label-cuts* efficiently by combining iterative expansions of cuts and the labeling in FlowMap [3]. The partial enumeration also enumerates other K-feasible cuts those are valid to minimize area of LUT network. Our algorithm can be summarized in figure 3. $\Omega_K(v_i)$ denotes the set of the cuts enumerated at v_i, and $\Psi(S)$ is the same with that in previous section, where S is a set of cuts. Our algorithm consists of two steps; labeling and cut enumeration. In the labeling, the label of each node in a subject graph is calculated. The label of each node is calculated with using labeling technique in FlowMap [3] without enumerating all cuts. The labels of all nodes can be calculated in $O(Kmn)$, where m and n denote the number of edges and the number of nodes in the subject graph, respectively. The details of labeling is omitted in this paper. In the cut enumeration, cuts for each node are enumerated in topological order from PI to PO. Cut enumeration for a node consists of two steps; *label-cut* enumeration and *area-cut* enumeration.

4.3.2 Label-Cut Enumeration

Let v be the current node being processed, and p be the label of v. The cuts of v can be divided into the two set of cuts; the set of cuts whose leaf includes any node with label p and the set of the others. A cut C of v whose leaf includes any node with label p is not a *label-cut* because $H(C)+1 > p$. Because the label of a node v cannot be smaller than that of any predecessor of v [3], the label of each node in $TFIG(v)$ are p or smaller than p. Thus, for a cut C' of v whose leaf does not include any node with label p, the label of each node in $LEAF(C')$ is smaller then p, and $H(C')+1 \leq p$. Then $H(C')+1 = p$ because the label of v is p. Thus, the set of cuts of v which do not include any node with label p is the set of *label-cuts* of v. A cut C is a *label-cut* if and only if the condition $\forall i \in LEAF(C), L_K(i) \neq p$ is satisfied. According to above discussion, cuts whose leaf includes no node with label p are enumerated. Thus, expansions of cuts C, where $\exists i \in LEAF(C), L_K(i) = p$, are not needed. After temporarily collapsing the nodes with label p , together with v into v', *label-cuts* of v can be enumerated with executing iterative expansions of cuts from a cut $(v', \{v'\})$. An example is shown in figure 1. To enumerate *label-cuts* of the node a, expansions $EXP(b, (a, \{b, c\}))$, $EXP(c, (a, \{b, c\}))$, $EXP(c, (a, \{c, d, e\}))$, and so on can be skipped. Iterative expansions of cuts can be started with the cut $(a, \{d, h, i, j\})$. Although there are 18 cuts in figure 1, the cuts $(a, \{d, h, i, j\})$ and $EXP(d, (a, \{d, h, i, j\}))$ are checked to enumerate *label-cuts* of a.

The expansions of cuts $EXP(i, C)$ for $i \notin FTP(v)$ are not executed in *label-cuts* enumeration, where $FTP(v) = \{w | w \in \{v\} \cup \Psi(\Omega_K(v_1)) \cup \Psi(\Omega_K(v_2))\}$, and $\{v_1, v_2\} = FI(v)$. Our algorithm cannot guarantee to enumerate all the *label-cuts* for v because $FTP(v)$ is the subset of $\{v\} \cup \Psi(\Phi_K(v_1)) \cup \Psi(\Phi_K(v_2))$. If there are no *label-cut* of v obtained, iterative expansions of cuts are executed again until all the nodes in leaf are in PI or a constant number of *label-cut* is obtained. This is the process to guarantee that there is at least several *label-cuts* for each node.

4.3.3 Depth-Optimality of Network derived with Label-Cuts

This subsection shows that the LUT network N derived by a realizable set S of *label-cuts* is depth-optimum.

For a *label-cut* $C \in S$ where $LEAF(C) \subseteq PI$, $KFC(C)$ clearly covers $TFIG(RT(C))$ in a depth-optimum way. For a generic *label-cut* $C \in S$, assume that $TFIG(i)$ for each $i \in CFI(C, S)$ is covered in depth-optimum way. Then, the subgraph of N which covers $TFIG(RT(C))$ is also depth-optimum. According to above two conditions, for each node v in subject graph where $\exists C \in S, v = RT(C)$, in topological order from PI to PO, $TFIG(v)$ is proved to be covered in a depth-optimum way. For all the $C \in S$, the subgraph of N which covers $TFIG(RT(C))$ is depth-optimum. Therefore, N is a depth-optimum LUT network for the subject graph.

4.3.4 Area-Cut Enumeration

An event that a node is covered with multiple K-feasible cones is called a node duplication. While node duplications are often necessary to minimize the depth of LUT network, unprofitable node duplications increase area. Depth-noncritical region in subject graph is not forced to be covered with *label-cuts* for depth-minimization. Covering depth-noncritical region with *label-cuts* may cause unprofitable node duplication. At *area-cut* enumeration step, cuts which may be favorable for area-minimization are enumerated in a heuristic technique. Practically, Intuitively, node v with many fanouts is likely to cause node duplication. If all of the fanouts are covered with one K-feasible cone, v is not duplicated. On the other hand, if the fanouts are covered with multiple K-feasible cones, and if any fanout is covered with a K-feasible cone C whose inputs do not include v, i.e. $v \notin INPUT(C)$, v must be duplicated. At this step, cuts whose all nodes in leaf have multiple fanouts are enumerated. Such cuts is called *area-cut* in this section.

Enumeration of *area-cut* at node v starts at $(v, \{v\})$, and iterative expansions of cuts are executed. To pass over cuts whose leaf contains any node with single fanout, any node in leaf with single fanout is always forced to be expanded. For example in figure 1, iterative expansions of cuts begin with $(a, \{a\})$. Then, $EXP(a, (a, \{a\})) = (a, \{b, c\})$ is obtained. In this case, b and c are forced to be expanded because $FO(b) = 1$ and $FO(c) = 1$, and then $(a, \{d, e, f\})$ is obtained. Notice that $(a, \{d, e, c\})$ and $(a, \{b, e, f\})$ are not enumerated. Furthermore, d and f have single fanout. Thus, d and f are forced to be extended, and $(a, \{g, h, e, i, j\})$ is obtained. If each of leaf has multiple fanouts, iterative expansions of cuts are executed as described in section 4.1. This iterative expansions stop if all nodes in leaf have smaller label than root, because *label-cut* is already obtained.

5. EXPERIMENTAL RESULTS

Experiments are performed to compare exhaustive enumeration, partial enumeration, and bottom-up algorithm. Exhaustive enumeration and partial enumeration have been implemented as programs with using C++. The bottom-up algorithm is according to the algorithm to enumerate all cuts described in papers [1] [2]. A depth-minimum covering algorithm [7] is applied to generate an LUT network. The depth of LUT networks are the same if given subject graph and K is the same for each algorithm, because both exhaustive enumeration , partial enumeration and bottom-up algorithm are guarantee to generate depth-optimum LUT network if they combined with a depth-minimum covering algorithm. Area of LUT networks and total run-time are evaluated to compare the algorithms. Total run-time means the sum of run-time of cut enumeration and run-time of covering . Our experiments are carried out on a machine with $3.00GHz$ CPU and $16GB$ memory. The subject graphs are obtained by decomposing nodes of each circuit on MCNC benchmark set and ITC'99 benchmark set to the nodes whose number of inputs are up to 2.

Table 1 shows the comparison of area and run-time between exhaustive enumeration, partial enumeration and bottom-up algorithm for $K = 8, 9$. We omit the results for small circuits. The columns of "Bu", "E", and "P" denote bottom-up algorithm, exhaustive enumeration, and partial enumeration. The column of "All" denotes the methods to enumerate all cuts, i.e. Bu or E. The depth of LUT network is the same if the given subject graph and K is the same for each algorithm. Exhaustive enumeration runs about 3 times and 8 times faster than bottom-up algorithm for $K = 8$ and 9, respectively. The area of LUT network generated by

Table 1: A comparison of our methods and bottom-Up enumeration for depth-minimum technology mapping

Circuits	K=8								K=9							
	#LUT			Run Time (sec)					#LUT			Run Time (sec)				
	All	P	P/All	All(Bu)	All(E)	P	E/Bu	P/Bu	All	P	P/All	All(Bu)	All(E)	P	E/Bu	P/Bu
C3540	170	173	102%	55.3	19.0	5.5	34%	10%	159	160	101%	685.9	102.2	36.7	15%	5.4%
C5315	217	218	100%	230.6	53.9	20.0	23%	9%	205	205	100%	4014.3	252.2	92.9	6%	2.3%
C6288	261	261	100%	2825.0	524.2	317.7	19%	11%	247	247	100%	50943.9	3333.8	1852.4	7%	3.6%
C7552	334	334	100%	953.6	104.8	54.5	11%	6%	289	289	100%	20637.4	572.8	235.6	3%	1.1%
att15	243	248	102%	21.5	4.4	1.4	20%	6%	188	191	102%	169.5	14.1	3.7	8%	2.2%
att16	260	260	100%	7.3	2.7	1.6	37%	22%	235	235	100%	32.8	8.4	2.9	26%	8.9%
att21	1083	1087	100%	19.5	10.1	5.8	52%	30%	976	1006	103%	132.7	32.4	15.2	24%	11.4%
att22	157	156	99%	2.8	1.0	0.5	35%	18%	136	137	101%	18.1	2.6	1.0	15%	5.6%
att8	226	241	107%	2.5	1.2	0.7	47%	27%	187	196	105%	16.5	3.4	1.8	21%	10.9%
des	369	369	100%	363.6	67.8	34.6	19%	10%	550	550	100%	16674.2	529.0	197.2	3%	1.2%
rot	198	194	98%	4.4	2.9	1.5	65%	34%	160	161	101%	33.5	10.6	4.6	32%	13.6%
b12	203	203	100%	9.0	4.0	3.0	44%	33%	173	175	101%	101.9	16.8	9.7	16%	9.6%
b14	1115	1190	107%	633.1	271.9	142.7	43%	23%	999	1081	108%	9489.1	1444.1	660.0	15%	7.0%
b14_1	958	1007	105%	513.8	162.1	97.8	32%	19%	906	943	104%	6989.9	758.5	396.5	11%	5.7%
b15	1578	1675	106%	1717.9	625.4	297.4	36%	17%	1498	1558	104%	40604.3	4160.1	1764.6	10%	4.3%
b15_1	1603	1802	112%	10403.9	702.5	555.0	7%	5%	1459	1566	107%	134717.9	3404.2	2335.6	3%	1.7%
b17	5091	5553	109%	12073.4	1653.5	1017.7	14%	8%	4751	5018	106%	179990.2	9649.6	4751.0	5%	2.6%
b17_1	4949	5559	112%	31814.1	2291.7	1740.5	7%	5%	4552	4980	109%	417510.6	13702.9	5621.9	3%	1.3%
b20	2151	2352	109%	1445.5	579.9	326.2	40%	23%	1969	2170	110%	20569.8	3002.6	1376.6	15%	6.7%
b20_1	1894	2008	106%	1387.0	407.3	235.3	29%	17%	1743	1797	103%	20313.5	2044.4	973.4	10%	4.8%
b21	2253	2467	109%	1339.4	563.9	319.0	42%	24%	2032	2270	112%	18410.0	2894.3	1323.9	16%	7.2%
b21_1	1966	2098	107%	1592.9	457.2	267.7	29%	17%	1825	1912	105%	23659.5	2233.5	1122.3	9%	4.7%
b22	3415	3652	107%	2437.9	972.5	553.9	40%	23%	2989	3236	108%	37582.8	5272.4	2621.7	14%	7.0%
b22_1	2885	3050	106%	2245.4	656.6	382.0	29%	17%	2635	2705	103%	32448.2	3216.4	1577.7	10%	4.9%
Average			104%				31%	17%			104%				12%	5.6%

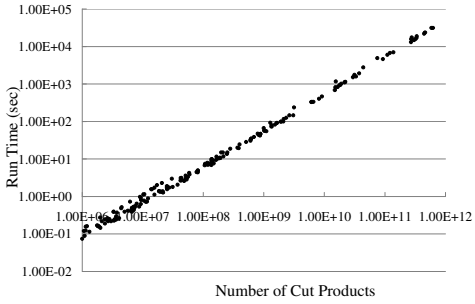

Figure 4: The run-time of bottom-up algorithm and the number of *cut products*

bottom-up algorithm and that by exhaustive enumeration are the same. Partial enumeration runs about 6 times and 18 times faster than bottom-up algorithm for $K = 8$ and 9, respectively, The area of LUT network generated by partial enumeration is only 4% larger than that generated by bottom-up algorithm. Exhaustive enumeration and partial enumeration are faster for $K = 9$ than the case for $K = 8$.

Figure 4 shows the run-time of bottom-up algorithm and the number of *cut products* with logarithmic axes. The number of *cut products* means the total number of elements of the sets (not only cuts) computed by the operations of *cut-merging*. Let V denotes the set of all nodes in subject graph. The number of *cut products* is calculated by $\sum_{v \in V} \mid \Phi_K(u_1) \mid \times \mid \Phi_K(u_2) \mid$, where $\{u_1, u_2\} = FI(v)$. Almost all the points lie on the diagonal line in figure 4. This results show that the computing complexity of bottom-up algorithm is proportional to number of *cut products* and does not depend on the number of cuts. Bottom-up algorithm is very inefficient in enumerating cuts because number of *cut products* is much larger than the number of cuts. For example in "b17_1", the number of *cut products* is about 5200 times larger than that of cuts.

6. CONCLUSIONS

This paper presents an efficient algorithm for cut enumeration to generate depth-minimum LUT networks for technology mapping for LUT-based FPGAs. The proposed algorithm narrows down the search range compactly by setting the condition to stop expansions of cuts appropriately. The proposed algorithm has two modes; exhaustive enumeration and partial enumeration. The exhaustive enumeration enumerates all of cuts. The partial enumeration enumerates limited cuts while it guarantees that any depth-minimum LUT network can be constructed. The experimental results show that exhaustive enumeration enumerates all cuts about 3 times and 8 times faster than bottom-up algorithm for $K = 8, 9$, respectively. If covering algorithms are the same, the quality of LUT network is the same. The experimental results also show that partial enumeration runs about 6 times and 18 times faster than bottom-up enumeration for $K = 8, 9$, respectively. The depth of LUT network generated by either of them is optimum. The area of LUT network derived by partial enumeration is only 4% larger than that derived by bottom-up algorithm. Exhaustive enumeration has an advantage over bottom-up algorithm in run-time. If a little penalty of area is acceptable, partial enumeration is more valid to run fast.

7. ACKNOWLEDGMENTS

This work has been partially supported by CREST-DVLSI of JST and Grant-in-Aid for Scientific Research (B) (20300020) of Ministry of Education, Culture, Sports, Science and Technology (MEXT). We are grateful for their support.

8. REFERENCES

[1] J.Cong, C.Wu, and Y.Ding, "Cut ranking and pruning: Enabling a general and efficient fpga mapping solution," in *Proc. FPGA '99*, 1999, pp. 29–35.

[2] D.Chen and J.Cong, "Daomap: A depth-optimal area optimization mapping algorithm for fpga designs," in *Proc. ICCAD '04*, 2004, pp. 752–759.

[3] J.Cong and Y.Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs," *IEEE Trans. CAD*, vol. 13, pp. 1–12, 1994.

[4] J.Cong and Y.Ding, "On area/depth trade-off in

lut-based fpga technology mapping," *IEEE Trans. on VLSI Systems*, vol. 2, pp. 213–218, 1994.

[5] J.Cong and Y.yow Hwang, "Simultaneous depth and area minimization in lut-based fpga mapping," in *Proc. ACM 3rd Int'l Symp. on FPGA*, 1995, pp. 68–74.

[6] M.Teslenko and E.Dubrova, "Hermes: Lut fpga technology mapping algorithm for area minimization with optimum depth," in *Proc. ICCAD '04*, 2004, pp. 748–751.

[7] T.Takata and Y.Matsunaga, "Area recovery under depth constraint by cut substitution for technology mapping for lut-based fpgas," in *Proc. ASP-DAC '08*, 2008, pp. 144–147.

[8] S.Chatterjee, A.Mishchenko, and R.Brayton, "Factor cuts," in *Proc. ICCAD '06*, 2006, pp. 143–150.

[9] A.Mishchenko, S.Cho, S.Chatterjee, and R.Brayton, "Combinational and sequential mapping with priority cuts," in *Proc. ICCAD '07*, 2007, pp. 354–361.

[10] P.Pan and C.-C.Lin, "A new retiming-based technology mapping algorithm for lut-based fpgas," in *Proc. FPGA '98*, 1998, pp. 35–42.