Optimal Clock Period FPGA Technology Mapping for Sequential Circuits

PEICHEN PAN
Clarkson University, Potsdam, NY
and
C. L. LIU
University of Illinois at Urbana-Champaign, Urbana, IL

We study the technology mapping problem for sequential circuits for look-up table (LUT) based field programmable gate arrays (FPGAs). Existing approaches to the problem simply remove the flip-flops (FFs), then map the remaining combinational logic, and finally put the FFs back. These approaches ignore the sequential nature of a circuit and assume the positions of the FFs are fixed. However, FFs in a sequential circuit can be repositioned by a functionality-preserving transformation called retiming. As a result, existing approaches can only consider a very small portion of the available solution space. We propose in this paper a novel approach to the technology mapping problem. In our approach, retiming is integrated into the technology mapping process so as to consider the full solution space. We then present a polynomial technology mapping algorithm that, for a given circuit, produces a mapping solution with the minimum clock period among all possible ways of retiming. The effectiveness of the algorithm is also demonstrated experimentally.

Categories and Subject Descriptors: B.6.1 [**Hardware**]: Design Styles—*sequential circuits*; B.6.3 [**Hardware**]: Design Aids—*automatic synthesis*; *optimization*

General Terms: algorithms, performance

Additional Key Words and Phrases: Clock period, field–programmable gate arrays, FPGAs, logic replication, look–up tables, retiming, sequential synthesis, technology mapping

1. INTRODUCTION

FPGA has evolved rapidly to become an important ASIC technology. The most conspicuous features of FPGAs are: a low manufacturing cost for

The work was partially supported by the National Science Foundation under grants MIP-9222408 and MIP-9612184. An abridged version of this paper appeared in *Proceedings of the ACM/IEEE Design Automation Conference*, 1996.
Authors' addresses: Peichen Pan, Department of Electrical and Computer Engineering, Clarkson University, Potsdam, NY 13699; email: panp@clarkson.edu; C. L. Liu, Department of Computer Science, University of Illinois at Urbana–Champaign, Urbana, IL 61801; email: liucl@cs.uiuc.edu.

low–volume designs, a short design cycle, and reprogrammability. These features make FPGAs particularly attractive for such applications as design prototyping and hardware emulation.

In this article we consider the popular LUT-based FPGA architectures [Altera 1995; AT&T Microelectronics 1995; Xilinx 1993]. A LUT-based FPGA chip consists of an array of programmable logic blocks together with programmable interconnects. The core of a programmable logic block is a k-input LUT (k-LUT) that can implement any combinational logic with up to k inputs and a single output, where k is a positive integer ranging usually from 3 to 9. There are also a few flip-flops (FFs) in each programmable logic block that can be connected to the inputs and the output of the LUT to realize sequential behavior.

The technology mapping problem for LUT-based FPGAs is to produce, for a given circuit, an equivalent circuit comprised of k-LUTs. This problem has been studied extensively. However, almost all mapping algorithms designed were for combinational circuits. Mapping algorithms for combinational circuits have been proposed to target various optimization criteria: performance,[1] area,[2] routability [Bhat and Hill 1992; Schlag et al. 1994], and combinations of those [Cong and Ding 1994; Sawkar and Thomas 1992]. We mention here particularly FlowMap [Cong and Ding 1994]. Given a k-bounded combinational circuit, FlowMap produces a mapping solution with the minimum level.

Although most circuits are sequential in practice, there have been limited research efforts in technology mapping for sequential circuits [Murgai et al. 1993; Weinmann and Rosenstiel 1993]. Most existing approaches are based on mapping algorithms for combinational circuits. Specifically, all the FFs in a circuit are removed to obtain a combinational network. Then, the combinational network is mapped. Finally, the FFs are replaced. These approaches have two serious drawbacks: (1) they fail to consider signal dependencies across FF boundaries, and (2) they do not consider the possibility of exposing the combinational logic between the FFs in different ways. Note that the positions of the FFs in a sequential circuit are not fixed and can be changed by a technique called retiming [Leiserson and Saxe 1991]. As a result, existing approaches only consider a very small portion of the available solution space.

In this article, we propose a new approach to the technology mapping problem for sequential circuits. In this approach, technology mapping is carried out directly, without resorting to a mapping algorithm for combinational circuits. Since FFs are not removed there is no circuit segmentation, and signal dependencies across FF boundaries are exploited fully. Moreover, the FF positions are assumed to be dynamic in that they can be arbitrarily repositioned through retiming. We focus on the performance

[1][Cong and Ding 1993; Cong and Ding 1994; Francis et al. 1991; Mathur and Liu 1994; Sawkar and Thomas 1993; Woo 1991]
[2][Farrahi and Sarrafzadeh 1994; Francis et al. 1990; Francis et al. 1991; Karplus 1991; Murgai et al. 1990; Murgai et al. 1991; Woo 1991]

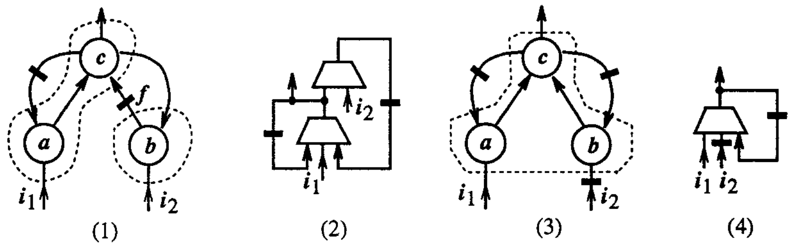

Fig. 1. Advantage of combining technology mapping with retiming.

aspect of the problem. Our main objective is to obtain mapping solutions with a minimum clock period, which is defined as the maximum number of LUTs between any two successive FFs.[3] We present a polynomial technology mapping algorithm that produces minimum clock period mapping solutions.

1.1 Motivating Examples

This example shows the advantage of repositioning FFs in technology mapping. Consider the circuit in Fig. 1a and assume $k = 3$. One possible mapping solution, without repositioning the FFs, is shown in Fig. 1b, where each LUT consists of the gates enclosed by a dashed circle in Fig. 1a. This mapping solution uses two LUTs and has a clock period equal to two. Retiming at this stage cannot reduce the clock period of this mapping solution since there is a cycle with two LUTs but only one FF. In fact, it can be shown that without retiming, any mapping solution of this circuit contains at least two LUTs and has a clock period at least two, no matter how the combinational logic is mapped. On the other hand, if the FF f at the output of gate b is moved to the inputs of b as shown in Fig. 1c, all the gates can be mapped to a single 3-LUT to form the mapping solution in Fig. 1d, which only has a clock period of one.

We further notice that to fully exploit the potential of retiming, *logic replication* is necessary. Replication can help produce mapping solutions which are impossible to obtain otherwise. Consider the circuit in Fig. 2a with $k = 4$. It can be shown that any mapping solution must use at least six 4-LUTs and must have a clock period at least two, even with retiming. However, if we duplicate a (to become a and a'), b (to become b and b'), and c (to become c and c') and then retime the FFs across gates a', b', and c' as shown in Fig. 2b, we can map all the gates (including the duplicated ones) to a single 4-LUT to obtain the mapping solution in Fig. 2c. This mapping solution has a clock period of one.

It should be noted that logic replication as used here has a purpose different from that in technology mapping for combinational circuits. As can be seen from the example in Fig. 2, replication is needed in forming just

[3]The unit-delay model is used here in which interconnect delays are ignored. The algorithm in this paper can be generalized to work for the general delay model. The generalized algorithm can produce mapping solutions with clock periods provably close to minimum.

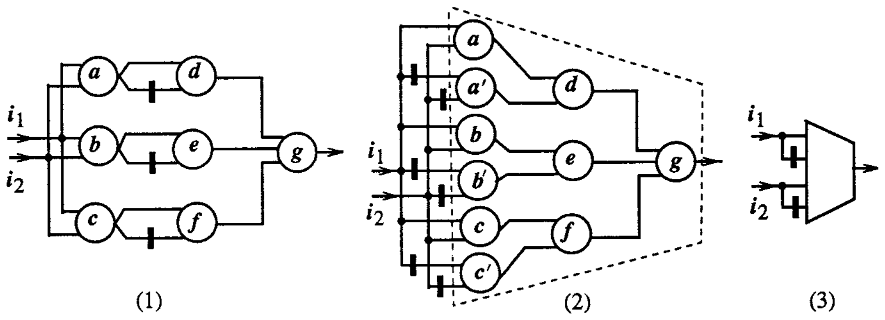

Fig. 2. Advantage of combining technology mapping with logic replication.

a single LUT. This is not necessary for combinational circuits, where replication means simply logic overlapping between different LUTs.

2. OVERVIEW OF THE NEW APPROACH

We examine the technology mapping problem in the general setting in which both retiming and replication are considered in this article. It is obvious that by incorporating retiming and replication the solution space becomes enormous as there are too many ways to retime and replicate a circuit.

Our problem formulation can be illustrated by the diagram in Fig. 3. That is, the solution space consists of all the circuits that can be obtained by retiming and replicating the circuit to be mapped, mapping the logic between the FFs, and then another retiming and replication.[4] Note that in existing approaches, Steps 1 and 3 in Fig. 3 are missing. As a result, existing approaches only consider a much smaller solution space than the one shown here.

We should emphasize that Fig. 3 shows the potential solution space that can be explored by our approach, and *not* the steps that our algorithm will take to solve the problem. In fact, mapping algorithms based on existing approaches may try to exploit retiming by carrying out these conceptual steps in sequence. However, such algorithms most likely arrive at suboptimal solutions, since it is impossible to foresee which retiming and replication to use in Step 1 before actually mapping the combinational logic. Moreover, the best solution to the combinational logic of a sequential circuit may not lead to the best solution to the sequential circuit [Pan 1997].

The algorithm we propose, on the other hand, finds the best mapping solution in the solution space shown in Fig. 3 without actually resorting to mapping algorithms for combinational logic. The algorithm operates di-

[4]Theoretically speaking, the retiming and replication after mapping the combinational logic can be merged with the one before the mapping. However, there are two reasons for having them separated: One is to make the formulation more general. The other is to allow the flexibility of having several steps of retiming. In fact, separating retiming for different purposes is one of the contributions of this paper.

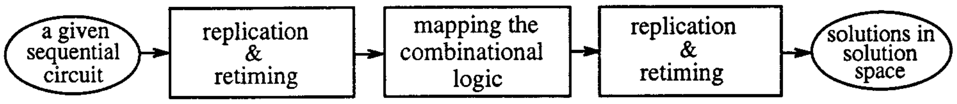
a given sequential circuit
replication & retiming
mapping the combinational logic
replication & retiming
solutions in solution space

Fig. 3. Solution space after integrating retiming and replication.

rectly on the sequential circuit. It seamlessly and globally integrates retiming, replication, and logic mapping to obtain mapping solutions with the ultimately best clock periods.

As we mentioned earlier, a mapping solution is a circuit comprised of LUTs. As a result, we need to address the following two issues:

—How to form LUTs for nodes in a sequential circuit in the presence of retiming and replication; and

—How to choose a "best" LUT for each node from among all possible ones to form a mapping solution.

Although LUT formation is rather simple for combinational circuits, it is much more complicated for sequential circuits because of the presence of retiming and replication. In fact, we no longer have a fixed circuit to work on. Instead, there is a family of circuits: all circuits that can be obtained by retiming and replicating the initial one. We derive a method for forming LUTs in sequential circuits based on *expanded circuits*, an important concept introduced in this article.

Our algorithm is based on dynamic programming in which a labeling scheme is used to guide the selection of LUTs to include in the final mapping solutions. We need a labeling scheme that takes into consideration both the number and the positions of the FFs, so we introduce a labeling scheme in which labels are tied to the minimum clock period of the mapping solutions. The final solution of the problem is based on repeated network flow computation.

The remainder of this article is organized as follows: In Section 3, we introduce some preliminaries and give a precise description of the problem. In Section 4, we discuss LUT formation in sequential circuits. In Section 5, we present a technology mapping algorithm for a target clock period. Section 6 describes the algorithm for finding a mapping solution with minimum clock period. Section 7 shows our experimental results. Finally, Section 8 concludes this article.

3. PRELIMINARIES AND PROBLEM DEFINITION

A (sequential) circuit can be modeled as an edge-weighted directed (multi-)graph. The nodes are the primary inputs (PIs), the primary outputs (POs), and the combinational processing elements (PEs). (A PE is either a gate or a k-LUT depending on whether the circuit is unmapped or a mapping solution.) The edges represent interconnections. There is an edge e from u to v (denoted $u \overset{e}{\rightarrow} v$) with weight t if the output of u, after passing through t FFs, is an input to v. We use N to denote the circuit to be mapped and

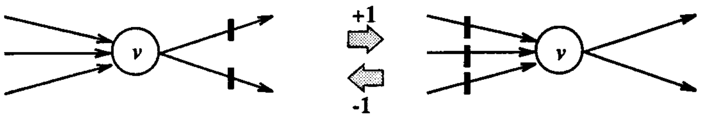

Fig. 4. Retiming a node.

$w(e)$ to denote the weight of edge e in N. We assume N is *k-bounded*, i.e., each gate has at most k inputs. We also assume that every node in N can reach at least one PO and can be reached from at least one PI.

Retiming is a transformation that relocates the FFs in a circuit without changing its functionality or structure [Leiserson and Saxe 1991]. Retiming a node by a value i is an operation that removes i FFs from each fan-out edge and adds i FFs to each fan-in edge of the node. Essentially, this operation delays the output of the node by i clock cycles. Of course, if i is negative, the output of the node is actually advanced by $-i$ clock cycles. Fig. 4 shows the case where $i = 1$ and -1. In general, all nodes in a circuit can be retimed simultaneously (a retiming of the circuit). It has been shown that the retimed circuit and the original one have the same functionality if the PIs and POs are not retimed (i.e., the retiming values of the PIs and POs are zero).

A retiming r can be represented by a mapping from the nodes to integers, where $r(v)$ denotes the retiming value for node v. In the circuit retimed according to r, the weight of an edge $u \xrightarrow{e} v$ becomes $w(e) + r(v) - r(u)$.

The *clock period* of a circuit is the maximum delay on the combinational paths (paths without FFs) in the circuit. The *retimed clock period* is the minimum circuit clock period that can be obtained by retiming the circuit. Polynomial algorithms for computing the retimed circuit clock period and a corresponding retiming can be found in Leiserson and Saxe [1991].

Unlike retiming, which does not modify the structure of a circuit, logic replication, on the other hand, is a transformation that structurally modifies a circuit while preserving the functionality. Replicating a node means to create several copies of the node and distribute the fan-outs of the node among the copies. Fig. 5 illustrates the duplication of a node. Note that all the fan-in edges of the node are duplicated too.

To introduce the new formulation of the technology mapping problem, we first specify the mapping solutions in the new formulation. Refer again to Fig. 3. Let N' be a circuit obtained from N using replication and retiming. Let N'' be a mapping solution of the combinational logic of N' (i.e., a k-LUT covering of the combinational logic of N', see Cong and Ding [1994] for more details), and S be the circuit obtained by placing the FFs in N' back to N'' and then following this by another retiming and replication.[5] Then, S is a *mapping solution* of N in the new formulation. The performance-driven technology mapping problem we address is:

[5]Actually, replication at this stage turns out to be unnecessary, but we list it here for the sake of completeness.

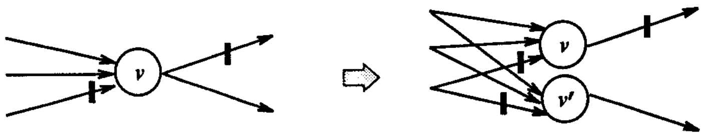

Fig. 5. Duplicating a node.

Problem 1. Find a mapping solution with minimum clock period.

Our approach to Problem 1 is based on solving the corresponding decision problem, which can be stated as follows:

Problem 2. Given a target clock period ϕ, find a mapping solution with a clock period less than or equal to ϕ, whenever such a mapping solution exists.

Obviously, with an algorithm for solving Problem 2, we can carry out binary search on the target clock period to find a mapping solution with a minimum clock period.

To conclude this section, we list several graph-theoretic concepts that we use later. In a directed acyclic graph (DAG) with one sink but possibly several sources, a *cut* $(X, \bar{X})$ is a partition of the nodes such that the sink is in X and all sources are in $\bar{X}$. The *edge-set* of the cut is the set of edges from $\bar{X}$ to X, the *node-set* of the cut is the set of nodes in $\bar{X}$ that are connected to one or more nodes in X. The *cone* of the cut is the subgraph induced by X. For ease of discussion, we denote cut $(X, \bar{X})$ by $[X]$, as $\bar{X}$ is usually understood from the context. We use $E[X]$ and $V[X]$ to denote the edge-set and node-set of $[X]$, respectively. If $|V[X]| \leq k$, $[X]$ is then called a k-cut.

4. FORMATION OF LUTS

In this section, we present a method for forming k-LUTs in a sequential circuit. Recall that in the case of combinational circuits, a k-LUT for a node is formed by the cone of a k-cut in the subcircuit for the node [Cong and Ding 1994]. However, in our problem formulation, a circuit may be retimed and replicated. As a result, there is no fixed circuit to use during LUT formation.

Due to the inclusion of retiming, the output of a LUT in a mapping solution may be retimed by a non-zero value (with respect to the corresponding node in the initial circuit). For example, in the mapping solution of the circuit in Fig. 6a, shown in Fig. 6c, the output of LUT $\mathcal{L}_a$ is retimed by 1, and the output of LUT $\mathcal{L}_c$ is retimed by -1. *A mapping solution in which the output of each LUT has a retiming value equal to zero is referred to as a simple mapping solution.* As an example, the circuit in Fig. 6b is a simple mapping solution of the circuit in Fig. 6a, since the outputs of LUTs $\mathcal{L}_a$ and $\mathcal{L}_c$ have retiming values equal to zero.

Note that there are still FF movements in a simple mapping solution. For example, gate d is retimed by a value of 1 in forming the simple mapping solution of Fig. 6b. The FF movement is used to form the LUT $\mathcal{L}_c$ for c.

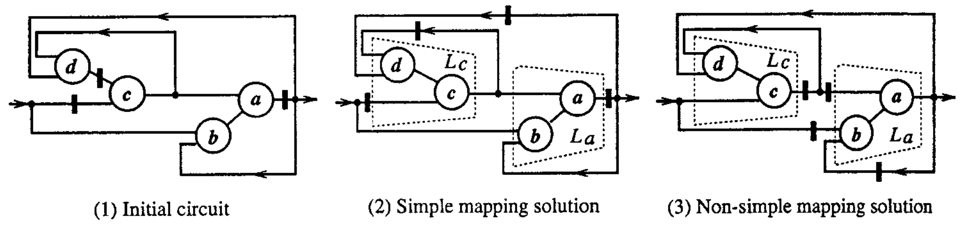

Fig. 6. Simple and non-simple mapping solutions: a. initial circuit; b. simple mapping solution; c. non-simple mapping solution.

THEOREM 1. *There is a mapping solution with a clock period less than or equal to ϕ iff there is a simple mapping solution with a retimed clock period less than or equal to ϕ.*

PROOF. (*if*) This part is obvious, since a retimed circuit of a mapping solution is itself a mapping solution, and a simple mapping solution with a retimed clock period of ϕ or less can, by definition, be retimed to a clock period of ϕ or less.

(*only if*) Suppose S is a mapping solution with a clock period of ϕ or less. We define a retiming r on S as follows: $r(\mathcal{L}) = -i$ for a LUT $\mathcal{L}$ in S, where i is the retiming value at the output of $\mathcal{L}$. That is, r takes the opposite retiming value of the output of $\mathcal{L}$. This retiming will cancel out the retiming at the output nodes of the LUTs. In the retimed mapping solution S_r, the retiming value of the output node of each LUT becomes zero. By definition, S_r is a simple mapping solution of N. As an example, for the mapping solution in Fig. 6c, $r(\mathcal{L}_a) = -1$ and $r(\mathcal{L}_c) = 1$. If this retiming is applied to the solution, we obtain exact the simple mapping solution in Fig. 6b.

The retimed clock period of S_r is less than or equal to the clock period of S, since we can retime S_r back to S. Therefore, S_r has a retimed clock period less than or equal to ϕ. □

Obviously, moving FFs across LUTs has nothing to do with LUT formation, although they are needed for clock period reduction. In a simple mapping solution, there is no retiming across LUTs. In fact, the retiming defined in the proof of Theorem 1 is meant to cancel such FF movements. The concept of simple mapping solutions lets us separate retiming for the purpose of LUT formation from retiming for clock period minimization. The separation is important because these two kinds of retiming are handled differently. As we show later, retiming for LUT formation is handled using expanded circuits, and retiming for clock period reduction is handled using a labeling scheme.

As a result of Theorem 1, instead of studying Problem 2 we study the following equivalent problem:

Problem 3. Find a simple mapping solution with a retimed clock period less than or equal to ϕ, whenever there is one.

By a restriction to simple mapping solutions, we need only to study LUT formation for the nodes in N (not in retimed circuits of N). Furthermore, the input to a LUT can be specified in terms of the output of a node in N after passing through a certain number of FFs. If the output of a LUT for a node u after passing through d FFs is an input to a LUT for v, we say simply (u, d) is an input to the LUT for v. We use *input*($\mathscr{L}$) to denote the set of inputs to a LUT $\mathscr{L}$. In the remainder of this section, we discuss LUT formation for simple mapping solutions.

We now introduce the concept of expanded circuits as a way to consider retiming and replication during LUT formation. The expanded circuit for a node v is formed by properly replicating the nodes in N, starting with v and going backward toward PIs. It is constructed in such a way that all paths from any given node to the only output node have the same number of FFs. In the expanded circuit for v, a node is denoted by u^d if the node is a copy of node u in N, where the index d is the number of FFs from u^d to the only output node v^0. The intuition in the construction of the expanded circuit for v is as follows: For a path from u to v in N

$$(u=)u_t \xrightarrow{e_t} u_{t-1} \xrightarrow{e_{t-1}} \cdots \xrightarrow{e_3} u_2 \xrightarrow{e_2} u_1 \xrightarrow{e_1} u_0(=v);$$

replication is used to create a unique corresponding path in the expanded circuit as follows:

$$u_t^{d_t} \xrightarrow{e_t} u_{t-1}^{d_{t-1}} \xrightarrow{e_{t-1}} \cdots \xrightarrow{e_3} u_2^{d_2} \xrightarrow{e_2} u_1^{d_1} \xrightarrow{e_1} u_0^{d_0(=0)},$$

where $d_i = d_{i-1} + w(e_i)$ is the total number of FFs on the path from u_i to v in N, and $u_i^{d_i}$ is a copy of u_i, for $1 \le i \le t$.

Given a node v in N, the expanded circuit for v, denoted $\mathscr{E}_v$, is constructed recursively. We start with one node v^0 in $\mathscr{E}_v$, then repeatedly carry out *expansion* at non-PI nodes in $\mathscr{E}_v$ that do not have incoming edges. Let u^d be a node without incoming edges. An expansion at u^d is defined as follows: For each edge $x \xrightarrow{e} u$ in N, add node x^{d_1} where $d_1 = d + w(e)$, to $\mathscr{E}_v$ if the node is not yet in $\mathscr{E}_v$, and add an edge $x^{d_1} \to u^d$ with weight $w(e)$ to $\mathscr{E}$. For the circuit in Fig. 1a, Fig. 7 shows four expansions in the construction of the expanded circuit for node c. From 7b to 7e each one is obtained from the preceding one by an expansion at the shaded node.

The expanded circuit $\mathscr{E}_v$ is a DAG with one sink v^0. Due to the possible presence of cycles, $\mathscr{E}_v$ may have infinitely many nodes. This is obviously the case for node c in Fig. 1a, since expansion will continue at both a^1 and b^2 in Fig. 7e and the expanded circuit becomes repetitive. We eventually use finite subcircuits of $\mathscr{E}_v$. An important class of subcircuits consists of: $\mathscr{E}_v^i$, for $i \ge 0$. $\mathscr{E}_v^i$ is comprised of the nodes in $\mathscr{E}_v$ whose shortest distance (in terms of the number of edges) to v^0 is less than or equal to i. For example, the circuit in Fig. 7a is $\mathscr{E}_v^0$, in 7b is $\mathscr{E}_v^1$, in 7d is $\mathscr{E}_v^2$, and in 7e is $\mathscr{E}_v^3$.

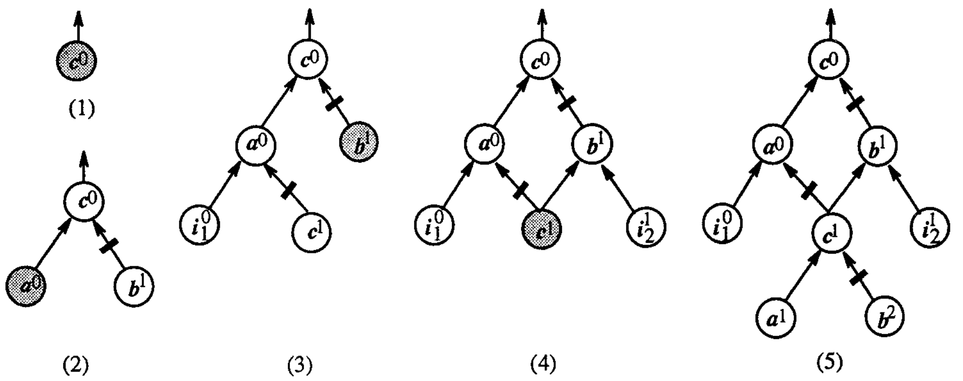

Fig. 7. Construction of an expanded circuit.

Let $[X]$ be a k-cut in the expanded circuit $\mathscr{C}_v$ for v. We can push the FFs in X to the edge-set of the cut. Specifically, we define a retiming on $\mathscr{C}_v$ that assigns a retiming value of d to each node u^d in X and zero to the rest of the nodes. It can be easily verified that, after the retiming, none of the edges between the nodes in X have FFs, and the number of FFs on an edge in $E[X]$ emanating from u^d is exactly d. From our problem definition, we know that the cone of this cut (after the retiming) is a LUT for v, and (u, d) is an input to the LUT for each u^d in $V[X]$ (so $[X]$ is actually a k-LUT). As an example, for the 4-cut with $X = \{c^0, c^1, b^1\}$ in the expanded circuit for c in Fig. 1a, shown in Fig. 8a, the corresponding 4-LUT is shown in Fig. 8b.

From the above discussion, we know that a k-LUT can be derived from a k-cut in an expanded circuit. The reverse of this statement is also true, as stated in the following result:

LEMMA 1. *Given a k-cut in $\mathscr{C}_v$, a k-LUT for v can be derived. The logic of the LUT is the cone of the cut less the FFs. (u, d) is an input to the LUT if u^d is in the node-set of the cut. On the other hand, given a k-LUT for v, there exists a k-cut in $\mathscr{C}_v$ that derives a k-LUT with the same set of inputs.*

PROOF. We have already shown the first part of this result in the preceding discussion. We now prove the second part, i.e., for any k-LUT, there is a k-cut in $\mathscr{C}_v$ that derives a k-LUT with the same inputs.

Let $\mathscr{L}$ be a k-LUT for v. By definition, $\mathscr{L}$ is the cone of a k-cut $\mathscr{C}$ in a circuit N'_v obtained from the subcircuit for v by replication and retiming. We first show that if we remove all edges directed out of $u_1^{d_1}$ for each (u_1, d_1) in *input*($\mathscr{L}$), all paths from the sources to v^0 will be broken in $\mathscr{C}_v$.

Let p be a path from a source to v^0 in $\mathscr{C}_v$. If we do not differentiate copies of the same node, p is also a path in N'_v, since replication does not introduce new paths. p must be broken by $\mathscr{C}$. That is, there is an edge $u \xrightarrow{e} x$ in both p and the edge-set of $\mathscr{C}$. Let d denote the number of FFs on e in N'_v. Since there is no FF on the edges inside the cone of $\mathscr{C}$ and neither u nor v is retimed (in a simple mapping solution), d must be equal to the total

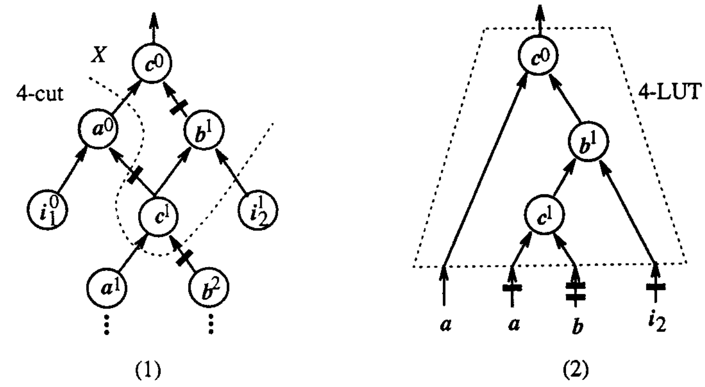

Fig. 8. Derivation of a LUT from a cut.

number of FFs on the segment of p from u to v. Thus, (u, d) is in *input*($\mathcal{L}$) and u^d is in $\mathcal{E}_v$. Hence, p is broken after the removal of the edges emitting from u^d.

Let X be the set of nodes in $\mathcal{E}_v$ that still have paths to v^0 after the removal of the edges. $[X]$ is a cut in $\mathcal{E}_v$, and the node-set of $[X]$ is $\{u_1^{d_1} \mid (u_1, d_1) \in$ *input*($\mathcal{L}$)$\}$. Hence, the input set of the k-LUT derived from cut $[X]$ is equal to *input*($\mathcal{L}$). □

Lemma 1 guarantees we can derive k-LUTs from k-cuts in expanded circuits. We now set out to show that the k-cuts are not "buried" too deep, so that we only need to examine a small finite portion of an expanded circuit to look for all k-cuts.

We first note that not all nodes in the node-set of a cut generate inputs useful to the corresponding LUT. For example, for the 4-cut formed by $X_1 = \{c^0, c^1\}$ in the expanded circuit shown in Fig. 8a, the node-set $V[X_1] = \{a^0, a^1, b^1, b^2\}$. However, neither a^1 nor b^2 generates useful input to the corresponding LUT, as they only drive gate c^1, which does not fan-out in the LUT. In this example, we can get rid of the redundant nodes a^1 and b^2 from the node-set by removing c^1 from X_1. In general, redundant nodes in the node-set of a cut $[X]$ can be removed by deleting all nodes in X that do not have a path to the output node in the cone of $[X]$. In the rest of this section, we assume redundant nodes in node-sets are removed.

LEMMA 2. *Let $[X]$ be a k-cut in $\mathcal{E}_v$; then for any node in the node-set $V[X]$, the minimum number of edges on the paths from the node to v^0 is at most kn.*

PROOF. By contradiction. Suppose there is a node y^h in $V[X]$ with the minimum number of edges larger than or equal to $kn + 1$. Since there is no redundant node in $V[X]$, there exists a node z in X such that $y^h \overset{e}{\rightarrow} z$ is in $E[X]$, and there is a path p from z to v^0 that lies completely in X. p must contain at least $kn + 1$ nodes. As a result, there is a node u for which at least $k + 1$ copies appeared in p. Let the copies be $u^{i_1}, u^{i_2}, \cdots, u^{i_t}$, where

$i_1 < i_2 < \cdots < i_t$ and $t \geq k + 1$. Let q be a simple path (a path with no repeated nodes) from a PI to u in N. Let $q^{i_1}, q^{i_2}, \cdots, q^{i_t}$ be the paths corresponding to q in $\mathscr{E}_v$, when u is replicated as $u^{i_1}, u^{i_2}, \cdots, u^{i_t}$, respectively. Obviously, q^{i_r} and q^{i_s} do not share any nodes for $r \neq s$. Since $[X]$ is a cut and p lies entirely in X, $E[X]$ must intersect with q^{i_s} for each $1 \leq s \leq t$. Consequently, there is a node u^{i_s} in both q^{i_s} and $V[X]$ for $1 \leq s \leq t$. Thus, $|V[X]| \geq t \geq k + 1$. This contradicts the assumption that $[X]$ is a k-cut. □

Combining Lemma 2 with Lemma 1, the main result of this section following:

THEOREM 2. *For any given $i \geq kn$, from a k-cut in $\mathscr{E}_v^i$, a k-LUT for v can be derived, and any k-LUT for v can be derived from a k-cut in $\mathscr{E}_v^i$.*

Finally, we estimate the numbers of nodes and edges in $\mathscr{E}_v^i$. We make the assumption that the number of FFs on any edge is at most one in N, as this is the case in practice. Since the shortest distance from u^d to v^0 is at most i, $d \leq i$ for any node u^d in $\mathscr{E}_v^i$. This implies u has at most $i + 1$ copies in $\mathscr{E}_v^i$. Hence, the number of nodes in $\mathscr{E}_v^i$ is $O(ni)$. Since each node in $\mathscr{E}_v^i$ has at most k inputs (because N is k-bounded), the number of edges in $\mathscr{E}_v^i$ is $O(kni)$. In particular, the numbers of nodes and edges in $\mathscr{E}_v^{kn}$ are $O(kn^2)$ and $O(k^2n^2)$, respectively. Of course, we expect both numbers to be much smaller in practice.

5. A MAPPING ALGORITHM FOR A TARGET CLOCK PERIOD

In this section, we present a polynomial algorithm for solving Problem 3: namely, finding a simple mapping solution with a retimed clock period less than or equal to ϕ. The algorithm has two phases: a labeling phase and a mapping phase. In the labeling phase, we find a label and an associated LUT for each node in N. We can also determine whether there are such mapping solutions from the labels. If the answer is positive, we then generate one in the mapping phase by connecting the LUTs obtained in the labeling phase. In the remainder of this section, we first introduce a labeling scheme which is used to guide the selection of LUTs. We then present the details of the two phases, separately.

5.1 A Labeling Scheme

Here we introduce a labeling scheme which is used to guide the selection of LUTs. For this, we first introduce the concept of l-values.

For each edge $\mathscr{L}_u \rightarrow \mathscr{L}_v$ in a mapping solution S, where $\mathscr{L}_v$ ($\mathscr{L}_u$) is a LUT for v (u), we assign an l-weight, $-\phi \cdot d + \delta(v)$, to it, where d is the number of FFs on the edge and $\delta(v)$ is the delay of $\mathscr{L}_v$ ($\delta(v) = 0$ if v is a PI or PO, and otherwise $\delta(v) = 1$).[6] The *l-value* of a LUT in S is defined as the maximum weight of the paths from the PIs to the LUT according to the

[6]Strictly speaking, the PIs and POs in a mapping solution are not LUTs. However, we will view them as LUTs formed by a single node, the node itself, for the convenience of discussion.

l-weights.[7] Note that some nodes may have infinite l-values if there are positive cycles in S. The importance of l-values is evident from the following result whose proof is in the appendix:

THEOREM 3. *S can be retimed to a clock period of ϕ or less iff the l-value of each PO is less than or equal to ϕ.*

Without loss of generality, we can assume from Theorem 3 that a mapping solution has at most one LUT for each node in N. Suppose a mapping solution has more than one LUT for a node v; we remove all LUTs for v except one with the smallest l-value. We then connect this LUT to all LUTs to which the removed LUTs for v were connected. Obviously, the l-values of the LUTs in the mapping solution will not be increased. From Theorem 3, the clock period will not be increased either.

The following result presents an upper bound on the l-value of each LUT in a mapping solution:

LEMMA 3. *If the l-value of a LUT for a node v in a mapping solution is finite, then the l-value is less than or equal to $-\phi \cdot w_v + n - 1$, where w_v denotes the minimum number of FFs on the paths from the PIs to v in N.*

PROOF. Let $\mathcal{L}_v$ be a LUT for v in a mapping solution S. We assume that S does not contain more than one LUT for each node in N. Thus, S has at most n nodes. Let p be a path in S from a PI to $\mathcal{L}_v$ with a total l-weight equal to the l-value of $\mathcal{L}_v$. p has at most $n - 1$ edges. The number of FFs on p is equal to the number of FFs on the corresponding path from the PI to v in N, which is larger than or equal to w_v. Therefore, the l-value of $\mathcal{L}_v$ is at most $-\phi \cdot w_v + n - 1$. $\square$

Using the concept of l-values, we now introduce the labeling scheme. The *label* of a node in N is simply the minimum of the l-values of the k-LUTs for the node in all simple mapping solutions, given ϕ as the target clock period.

5.2 The Labeling Phase

We determine the label and an associated k-LUT for each node in N in this phase of the algorithm. Due to the possible presence of cycles, we may not be able to determine the labels in one traversal of the circuit, since cycles obviously introduce cyclic dependencies among the labels.

As in a typical shortest path algorithm, we use dynamic programming to determine the labels. The approach is to maintain a lower bound on each label and then repeatedly improve the lower bounds until no improvement is possible, in which case the lower bounds have settled down to the labels. The lower bounds for the PIs are always zero. We set the initial lower bounds for all non-PI nodes to $-\infty$. Fig. 9 shows a high-level description of the algorithm, where $l(v)$ holds the current lower bound on the label of v.

[7]The concept of l-values is a special case of the continuous retiming introduced in Pan [1997].

```
L_FIND($N, \phi$)
1.  for each node $v$ in $N$ do    // initialization
2.      if ($v$ is a PI) $l(v) \leftarrow 0$;
3.      else $l(v) \leftarrow -\infty$;
4.  for $i \leftarrow 1$ to $\infty$ do
5.      settled $\leftarrow$ TRUE;
6.      for each non-PI node $v$ in $N$ do    // improving $l(v)$
7.          if IMPROVE($v$) = TRUE, settled $\leftarrow$ FALSE;
8.      if settled = TRUE, return SUCCESS;

IMPROVE($v$)
a.  Determine $l_{new}(v)$, the improved lower bound for $v$;
b.  if $l(v) < l_{new}(v)$
c.      $l(v) \leftarrow l_{new}(v)$;    // updating $l(v)$
d.      return TRUE;
e.  return FALSE;    // not updated
```

Fig. 9. The labeling procedure.

The actual improvement of the lower bounds is carried out by the procedure IMPROVE.

The procedure IMPROVE tests to see whether the lower bound for v has to be improved, based on the current lower bounds and, if so, updates the lower bound. In the procedure, $l_{new}(v)$ denotes the new lower bound for v computed from the current lower bounds in the circuit.

We now present a formula for $l_{new}(v)$. Suppose that $\mathcal{L}$ is a k-LUT for v. Remember that the l-weight of an edge to $\mathcal{L}$, with d FFs, is $-\phi \cdot d + \delta(v)$. Based on the current lower bounds, the l-value of $\mathcal{L}$ is at least

$$\max\{l(u) - \phi \cdot d + \delta(v) | (u, d) \in input(\mathcal{L})\}.$$

Since the label we are trying to compute is the minimum l-value of the k-LUTs for v, we would like to minimize $l_{new}(v)$. Thus, we use the following formula for $l_{new}(v)$:

$$l_{new}(v) = \min_{\mathcal{L}, \text{a } k\text{-LUT for } v} (\max\{l(u) - \phi \cdot d + \delta(v) | (u, d) \in input(\mathcal{L})\}). \tag{1}$$

With the above formula for $l_{new}(v)$, we now show that the labeling procedure indeed maintains lower bounds on the labels and approaches the labels progressively.

LEMMA 4. *For any node v in N, $l(v)$ is* (1) *non-decreasing, and* (2) *less than or equal to the l-value of any k-LUT for v in any mapping solution.*

PROOF. That $l(v)$ is non-decreasing is obvious from the procedure IMPROVE. We now show (2) using induction (on the number of calls to the procedure IMPROVE). The initial lower bounds obviously imply the statement before any call to IMPROVE. Now suppose that before the current call, the statement is true. Assume the current call does update $l(v)$. Right after the call, we have that $l(v) = l_{new}(v)$ and no change to the other lower bounds. Let $\mathcal{L}_v$ be a k-LUT (for v) in a mapping solution S. We only need to show that $l_{new}(v)$ is less than or equal to the l-value of $\mathcal{L}_v$ in S. Let l_u denote the l-value of the LUT for u in S; we have,

$$\begin{aligned} l(v) = l_{new}(v) &\le \max\{l(u) - \phi \cdot d + \delta(v) \mid (u, d) \in input(\mathcal{L}_v)\} \quad \text{(Eq. (1))} \\ &\le \max\{l_u - \phi \cdot d + \delta(v) \mid (u, d) \in input(\mathcal{L}_v)\} \\ &\qquad\qquad \text{(induction hypothesis)} \\ &= l_v \text{(definition of } l\text{-values).} \end{aligned}$$

□

For each node v in N, let $l^{opt}(v)$ denote the final $l(v)$ from the labeling procedure L_FIND. From Lemma 4, it is obvious that $l^{opt}(v)$ is less than or equal to the l-value of any k-LUT for v in any mapping solution. In fact, it can be shown that $l^{opt}(v)$ is equal to the label of v. (We do not include the proof as it is not needed for our purposes.) Note that each improved lower bound has an associated k-LUT that achieves the bound. A k-LUT associated with $l^{opt}(v)$ will be referred to as an *optimal* k-LUT for v. Optimal k-LUTs will be used to construct mapping solutions in the mapping phase. The following corollary of Lemma 4 is obvious:

COROLLARY 1. *If there is a PO v such that $l^{opt}(v) > \phi$, then N does not have a mapping solution with a retimed clock period less than or equal to ϕ.*

The lower bounds of some nodes may never settle down. In other words, their lower bounds can always be improved and can be arbitrarily large. If $l(v)$ can always be improved, we simply let $l^{opt}(v) = \infty$. Later, we will see that if there are infinite values, the target clock period cannot be achieved.

We now describe an approach to computing $l_{new}(v)$. It has been shown in Section 4 that the k-LUTs for v can be derived from k-cuts in $\mathcal{E}_v^{kn}$. We therefore have the following equivalent formula for $l_{new}(v)$:

$$l_{new}(v) = \min_{[X], \text{a } k\text{-cut in } \mathcal{E}_v^{kn}} (\max\{l(u) - \phi \cdot d + \delta(v) \mid u^d \text{ is in } V[X]\}). \quad (2)$$

To determine $l_{new}(v)$, we consider the following decision problem:

Problem 4. Determine whether $l_{new}(v) \le L$ for a given integer L.

We use network flow techniques to solve this problem. A flow network G is constructed from $\mathcal{E}_v^{kn}$ by applying to $\mathcal{E}_v^{kn}$ a standard network transformation called *node-splitting* in order to transform the problem of finding a k-cut to that of finding a cut with a capacity bound. To do so, each node in $\mathcal{E}_v^{kn}$, except v^0, is split into two nodes with a bridging edge between them. A super-source is added and connected to all the sources. The bridging edge

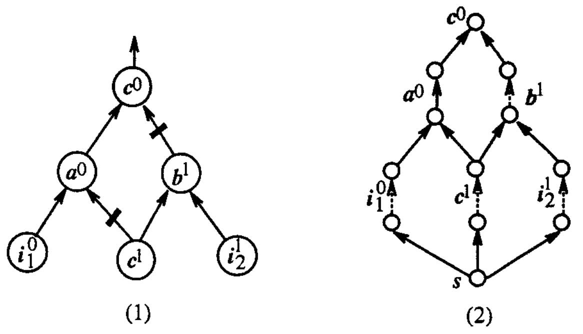

Fig. 10. Construction of flow network.

corresponding to node u^d has a capacity of one if $l(u) - \phi \cdot d + \delta(v) \leq L$. All other edges in G have an infinite capacity.

As an example, suppose for the circuit in Fig. 1a, we currently have $l(i_1) = l(i_2) = 0$, $l(a) = l(b) = 1$, and $l(c) = -\infty$, and the target clock period is one. We want to test whether $l_{new}(v) \leq 1$, using $\mathscr{C}_c^2$ shown in Fig. 10a to examine 3-cuts for c. For node b^1, $l(b) - \phi \cdot 1 + 1 = 1$, so the corresponding bridging edge has a capacity of one. On the other hand, for node a^0, $l(a) - \phi \cdot 0 + 1 = 2$, so the corresponding bridging edge has an infinite capacity. Fig. 10b shows the flow network, where the bridging edges for nodes i_1^0, i_2^1, c^1, and b^1 have unit capacity and all other edges have infinite capacity.

The capacity of a cut in a flow network is the sum of the capacities of the edges in the edge-set of the cut. We have the following result:

LEMMA 5. *$l_{new}(v) \leq L$ iff G has a cut with a capacity no more than k.*

PROOF. Suppose G has a cut $[X']$ with a capacity no more than k in G. Let X be the set of nodes consisting of v^0 and nodes in $\mathscr{C}_v^{kn}$ with both split nodes in X'. Obviously, a node u^d is in the node-set of cut $[X]$ iff the corresponding bridging edge is in the edge-set of cut $[X']$. As a result, $[X]$ is a k-cut in $\mathscr{C}_v^{kn}$. Moreover, $l(u) - \phi \cdot d + \delta(v) \leq L$ for each u^d in $V[X]$. Thus, $l_{new}(v) \leq \max\{l(u) - \phi \cdot d + \delta(v) | u^d \in V[X]\} \leq L$. The other direction can be shown by reversing the preceding arguments. We omit the details. □

Based on the classical Max-flow Min-cut theorem [Cormen et al. 1990], G has a cut with a capacity no more than k iff the maximum flow in G is at most k. Using an augmenting path algorithm for the max-flow problem, we can determine whether G has a cut with a capacity no more than k in $O(k \cdot |E(G)|) = O(k^3 n^2)$ time. Thus, we can determine whether $l_{new}(v) \leq L$ in $O(k^3 n^2)$ time.

Now $l_{new}(v)$ can be determined using binary search, since its potential values are: $l(u) - \phi \cdot d + \delta(v)$ for each u^d in $\mathscr{C}_v^{kn}$. However, the potential values for $l_{new}(v)$ can be narrowed down to two choices because we have the following result:

LEMMA 6. *$l_{new}(v) \geq L_v - 1$, where $L_v = \max\{l(u) - \phi \cdot w(e) + \delta(v) | u \xrightarrow{e} v$ is in $N\}$.*

PROOF. Let $u_1 \xrightarrow{e} v$ be an edge in N. It suffices to show $l_{new}(v) \geq l(u_1) - \phi \cdot d_1 + \delta(v) - 1$, where $d_1 = w(e)$.

Obviously, $(u_1^{d_1}, v^0)$ is an edge in $\mathscr{E}_v$. Let $[X]$ be a k-cut such that $l_{new}(v) = \max\{l(u) - \phi \cdot d + \delta(v) | u^d \in V[X]\}$. If $u_1^{d_1}$ is not in X, then $(u_1^{d_1}, v^0)$ must be in the edge-set of $[X]$, so $u_1^{d_1} \in V[X]$. As a result, $l_{new}(v) \geq l(u_1) - \phi \cdot d_1 + \delta(v)$. On the other hand, suppose $u_1^{d_1}$ is in X. Let X' (Y') be the set of nodes in X $(V[X])$ that have a path to $u_1^{d_1}$. Let $X_1 = \{u^{d-d_1} \mid u^d \in X'\}$ and $Y_1 = \{u^{d-d_1} \mid u^d \in Y'\}$. It can be verified that $[X_1]$ is a k-cut in $\mathscr{E}_{u_1}$ and Y_1 is the node-set of $[X_1]$. Therefore,

$$\begin{aligned} l(u_1) &\leq \max\{l(u) - \phi \cdot (d - d_1) + \delta(u_1) | u^{d-d_1} \in Y_1\} \\ &= \max\{l(u) - \phi \cdot d + \delta(v) | u^d \in Y'\} + \phi \cdot d_1 + \delta(u_1) - \delta(v) \\ &\leq \max\{l(u) - \phi \cdot d + \delta(v) | u^d \in V[X]\} + \phi \cdot d_1 + \delta(u_1) - \delta(v) \\ &= l_{new}(v) + \phi \cdot d_1 + \delta(u_1) - \delta(v) \end{aligned}$$

Thus, $l_{new}(v) \geq l(u_1) - \phi \cdot d_1 + \delta(v) - \delta(u_1) \geq l(u_1) - \phi \cdot d_1 + \delta(v) - 1$. □

In Eq. (2), by letting $X = \{v^0\}$, we have $l_{new}(v) \leq L_v$. Hence, to determine $l_{new}(v)$, we simply check to see whether $l_{new}(v) \leq L_v - 1$. If so, $l_{new}(v) = L_v - 1$; otherwise, $l_{new}(v) = L_v$.[8] As a result, $l_{new}(v)$ can be determined in time $O(k^3 n^2)$.

Another implication of Lemma 6 is that if there are unsettled lower bounds, the target clock period cannot be achieved. Let u be a node such that $l^{opt}(u) = \infty$. Let p be a path from u to a PO v. By applying Lemma 6 to all edges in p, we have $l(v) \geq l(u) - \phi \cdot \Sigma_{e \in p} w(e)$. Since $l(u)$ can be arbitrarily large, $l(v)$ can be arbitrarily large too. In particular, we have $l(v) > \phi$, which implies the clock period ϕ is unachievable.

We now estimate the time cost of the labeling procedure L_FIND, assuming the lower bounds will settle down. For that, we need to determine the number of iterations of the **for** loop. First, we show the following result:

LEMMA 7. *After $n - 1$ iterations, $l(v) \geq -\phi \cdot w_v$ for any node v.*

PROOF. By induction (on t), it is easy to show the following:

After t iterations, $l(v) \geq -\phi \cdot w_v$ for any v to which there is a path with w_v FFs and t edges. □

[8]Cong and Wu [1996] first observed that one test is enough to find an improved lower bound. They achieved that by forcing the improved bound to either $L_v - 1$ or L_v. In Lemma 6 we show the two values are in fact the only choices for the improved bound.

```
L_FIND($N, \phi$)
for each node $v$ in $N$ do // initialization
    $l(v) \leftarrow \infty$;
    if $v$ is a PI
        $l(v) \leftarrow 0$;
        $input(\mathcal{L}_v) \leftarrow \emptyset$;
        $\mathcal{L}_v \leftarrow v$; // The LUT for a PI is itself.
    if $v$ is a PO
        $\mathcal{L}_v \leftarrow v$; // The LUT for a PO is itself.
        $input(\mathcal{L}_v) \leftarrow \{(u, w(e))\}$; // $u \xrightarrow{e} v$ is the edge to $v$
for $i = 1$ to $n^2$ do
    settled $\leftarrow$ TRUE;
    for each node $v$ in $N$ do
        if $v$ is a PO
            $l(v) \leftarrow l(u) - \phi \cdot d$; // $input(\mathcal{L}_v) = \{(u, d)\}$
            if $(l(v) > \phi)$ return FAILURE;
        else
            $L_v \leftarrow \max\{l(u) - \phi \cdot w(e) + \delta(v) \mid u \xrightarrow{e} v \text{ is in } N\}$;
            $G \leftarrow$ the flow network constructed using $\mathcal{E}_v^{kn}$ with $L = L_v - 1$;
            if $G$ has a cut $[X']$ of capacity no more than $k$
                $l(v) \leftarrow L_v - 1$;
                $X \leftarrow \{v^0\} \cup \{u \mid$ both split nodes of $u$ are in $X'\}$;
                $\mathcal{L}_v \leftarrow$ the circuit induced by $X$ in $\mathcal{E}_v^{kn}$ (less the FFs);
                $input(\mathcal{L}_v) \leftarrow \{(u, d) \mid$ the bridging edge for $u^d$ is in $E[X']\}$
            else
                $l(v) \leftarrow L_v$;
                $\mathcal{L}_v \leftarrow \{v\}$;
                $input(\mathcal{L}_v) \leftarrow \{(u, w(e)) \mid u \xrightarrow{e} v \text{ is in } N\}$;
            if $l(v)$ has been changed, settled $\leftarrow$ FALSE;
    if settled = TRUE, return SUCCESS;
if settled = FALSE, return FAILURE;
```

Fig. 11. The labeling procedure—a more detailed version.

Combining Lemma 7 with Lemmas 3 and 4, we have that after $n - 1$ iterations, $l(v)$ is between $-\phi \cdot w_v$ and $-\phi \cdot w_v + n - 1$. For each additional iteration before the bounds settle down, at least one of the lower bounds is increased by at least one. Thus, after $n[(-\phi w_v + n - 1) - (-\phi w_v)] = n(n - 1)$ additional iterations, all lower bounds must reach their maximal possible values. In total, after $n(n - 1) + n - 1 = n^2 - 1$ iterations, all lower bounds should settle down. Of course, if improvements are still possible for some lower bounds in n^2-th iteration, L_FIND can simply stop since those unsettled lower bounds are arbitrarily large and the clock period cannot be achieved. Thus, we can set the number of iterations in L_FIND to be n^2. Obviously, this bound is grossly conservative. We observed that in practice usually the number of iterations is around ten, with a few further enhancements. With this bound on the number of iterations, L_FIND calls the procedure IMPROVE $O(n^3)$ times in the worst case. Thus, the time cost of L_FIND is $O(k^3 n^5)$. Fig. 11 is a more detailed description of L_FIND. In the procedure, a test is also added to check whether there is already a PO with a lower-bound larger than ϕ. If it is the case, the algorithm stops by returning FAILURE, as it is obvious that the clock period cannot be achieved.

The key results in this section are summarized in Theorem 4:

THEOREM 4. *(i) L_FIND returns SUCCESS iff $l^{opt}(v) \leq \phi$ for each PO v; (ii) if L_FIND returns SUCCESS, $l^{opt}(v)$ is less than or equal to the l-value of any LUT for v in any simple mapping solution; and (iii) if L_FIND returns FAILURE, there is no simple mapping solution with a retimed clock period less than or equal to ϕ.*

5.3 The Mapping Phase

The purpose of this phase is to generate a mapping solution with a clock period of ϕ or less. This phase is invoked only when the labeling procedure returns SUCCESS.

Remember that during the labeling phase, in addition to the labels, we also generate an optimal k-LUT for each node in N. The first step in the mapping phase is to assemble the optimal k-LUTs to form a simple mapping solution with a retimed clock period less than or equal to ϕ.

We start the assembly at the POs. For each node, its optimal k-LUT will be included in the simple mapping solution if the node (after passing through certain number of FFs) is an input to a k-LUT that has already been included in the solution. More precisely, we maintain a list of nodes U. U consists of the nodes whose optimal k-LUTs have already been included in the simple mapping solution, but the inputs to the k-LUTs have not yet been established. Initially, the simple mapping solution S consists of the PIs and POs, and U consists of the POs. At each iteration, a node v is removed from U. Let $\mathcal{L}_v$ be the optimal k-LUT for v. For each (u, d) in $input(\mathcal{L}_v)$, we first check to see whether $\mathcal{L}_u$ is already in S. If not, we add u to U and add $\mathcal{L}_u$, the optimal k-LUT for u, to S. Then, an edge with d FFs from $\mathcal{L}_u$ to $\mathcal{L}_v$ is created in S. This process continues until U becomes empty.

LEMMA 8. *For each k-LUT $\mathcal{L}_v$ in S, its l-value is less than or equal to $l^{opt}(v)$.*

PROOF. Take any edge $\mathcal{L}_u \rightarrow \mathcal{L}_v$ in S. Since $\mathcal{L}_v$ is an optimal k-LUT for v, $l^{opt}(v) = \max\{l^{opt}(u_1) - \phi \cdot d_1 + \delta(v) | (u_1, d_1) \in input(\mathcal{L}_v)\}$. Thus, $l^{opt}(v) \geq l^{opt}(u) - \phi \cdot d + \delta(v)$, where d is the number of FFs on the edge. After rewriting, we have,

$$l^{opt}(v) - l^{opt}(u) \geq -\phi \cdot d + \delta(v). \tag{3}$$

Now consider any path p from a PI to a LUT $\mathcal{L}_v$ in S. Applying Eq. (3) to all edges in p and adding all inequalities together, we have, $l^{opt}(v)$ is larger than or equal to the l-weight of p. Hence, $l^{opt}(v)$ is larger than or equal to the l-value of $\mathcal{L}_v$. □

From Lemma 8 and Theorem 3, we have that S can be retimed to a clock period of ϕ or less if $l^{opt}(v) \leq \phi$ for each PO v (or equivalently, L_FIND returns SUCCESS). On the other hand, from Theorem 4, if $l^{opt}(v) > \phi$ for some PO v (or equivalently, L_FIND returns FAILURE), N has no mapping

```
// Mapping phase
$U \leftarrow$ the set of POs;
$S \leftarrow \{\mathcal{L}_v \mid v$ is a PI or PO$\}$;
while $U \neq \emptyset$ do
    $v \leftarrow$ a node in $U$;
    $U \leftarrow U - \{v\}$;
    for each $u$ such that $(u, d) \in input(\mathcal{L}_v)$ do
        if $\mathcal{L}_u$ is not in $S$
            $S \leftarrow S \cup \{\mathcal{L}_u\}$;
            $U \leftarrow U \cup \{u\}$;
        create an edge in $S$ from $\mathcal{L}_u$ to $\mathcal{L}_v$ with $d$ FFs;
$S_r \leftarrow$ the circuit obtained by retiming $S$ according to $r$ in Eq. (4);
return $S_r$;
```

Fig. 12. The mapping procedure.

solution with a retimed clock period of ϕ or less. Therefore, we have the following result:

THEOREM 5. *The following statements are equivalent: (i) N has a mapping solution with a retimed clock period of ϕ or less; (ii) S can be retimed to a clock period of ϕ or less; and (iii) L_FIND returns SUCCESS.*

From Lemma 8 and the proof of Theorem 3, we can simply apply the following retiming to S to obtain a mapping solution with a clock period of ϕ or less:

$$r(\mathcal{L}_v) = \begin{cases} 0 & v \text{ is a PI or PO} \\ \left\lceil \dfrac{l^{opt}(v)}{\phi} \right\rceil - 1 & \text{otherwise.} \end{cases} \tag{4}$$

The mapping procedure is summarized in Fig. 12.

6. A MAPPING ALGORITHM FOR MINIMUM CLOCK PERIOD

We now summarize the algorithm for finding a mapping solution with the minimum clock period. The algorithm simply uses binary search to determine the minimum clock period by repeatedly calling the labeling procedure on different target clock periods.

In theory, we need to use $\mathscr{C}_v^{kn}$ in order to consider all k-cuts of a node v. This is the worst case scenario as the bound kn applies to all nodes in the circuit without considering any node specific information. In practice, it may be sufficient to use $\mathscr{C}_v^i$ for an i that is considerably smaller than kn. For instance, for node c in the circuit in Fig. 1a, all 3-cut are in $\mathscr{C}_c^2$ shown in Fig. 7d. In general, we only need to expand a node to the extent that further expansion will not introduce any new k-cuts. Recently, Cong and Wu [1996] proposed some techniques to reduce the sizes of subcircuits used in LUT formulation.

To make our algorithm flexible and to save computation time, the algorithm uses i as a controlling parameter. It only consider k-cuts in $\mathscr{C}_v^i$ in the procedure IMPROVE. Let L_FIND (N, ϕ, i) denote the modified L_FIND. Fig. 13 summarizes our algorithm for finding a mapping solution with a

```
SeqMap($N$)
$h \leftarrow$ a lower bound on the minimum clock period (e.g., 1);
$l \leftarrow$ an upper bound on the minimum clock period (e.g., n);
while ($h - l \geq 0$) {
    $\phi \leftarrow (l + h)/2$;
    if (L_FIND($N, \phi, i$) = SUCCESS) $h \leftarrow \phi$;
    else $l \leftarrow \phi + 1$;
}
Call the algorithm in the previous section with $\phi = h$;
```

Fig. 13. SeqMap: an optimal clock period mapping algorithm.

minimized clock period, which will be referred to as SeqMap. The time cost of SeqMap is $O(k^2n^4i \log n)$ in the worst case. If $i < kn$, the clock periods of the mapping solutions produced by SeqMap may not be optimal, although we believe that this situation occurs rarely. In practice, we may set i to be considerably smaller than kn and will still find optimal clock period mapping solutions.

There are many ways in which SeqMap can be enhanced, especially in L_FIND. It has been shown that, if the circuit contains no cycles such as pipelined circuits, the labels can be determined in one iteration by computing the labels in topological order starting from the PIs [Pan and Liu 1996]. One enhancement to L_FIND is to order the nodes and improve the lower bounds according to this order during each pass of improvement. The order should place the fan-ins of a node ahead of the node itself as much as possible. Our approach is to find a small feedback vertex set (a set of nodes whose removal breaks all cycles in the circuit) and generate a topological order by treating the nodes in the feedback set as additional primary inputs. We find this technique to be quite effective in reducing the run time of L_FIND. The run time of SeqMap can also be reduced by narrowing the ranges of the binary search for the target clock period. For example, we can set the upper bound of the minimum clock period to the clock period of a mapping solution produced by an existing algorithm.

7. EXPERIMENTAL RESULTS

We have implemented SeqMap in the C language and tested it on a set of benchmark circuits. In this section, we describe our experiments and summarize the results.

For comparison, we also implemented a technology mapping algorithm based on the conventional approach, which we refer to as ComMap. ComMap maps a sequential circuit by mapping the combinational logic between FFs using FlowMap—a delay–optimal technology mapping algorithm for combinational circuits [Cong and Ding 1994]. ComMap also uses retiming for pre- and post-processing. It retimes the initial circuit to its retimed clock period before applying FlowMap. ComMap also retimes the mapping solution to its retimed clock period after FlowMap. In our current implementation, neither ComMap nor SeqMap tries to reduce the number of LUTs, nor do they attempt to minimize the number of FFs in the final mapping solution.

Table I. Experimental Results

test circuit	Initial		ComMap			SeqMap		
	#gates	#FFs	#LUTs	#FFs	ϕ	#LUTs	#FFs	ϕ
ex1	326	20	202	56	6	209	60	5
ex5	105	9	72	29	4	58	24	3
mult16a	261	16	75	58	3	38	55	2
mult32a	533	32	153	202	3	78	207	2
s344	109	15	50	40	3	36	36	2
s349	112	15	49	39	3	33	33	2
s382	148	21	73	49	3	64	43	2
s400	158	21	72	47	3	66	46	2
s444	169	21	77	51	3	64	43	2
s526	252	21	166	84	3	127	89	2
s526n	251	21	166	85	3	140	97	2
s953	348	29	196	61	5	202	54	4
s1488	734	6	339	63	5	266	25	4
s9234	2352	193	590	270	5	593	276	4
s15850	3852	522	1670	704	9	1627	818	8
s38417	8709	1583	4170	2503	8	3761	2507	6
Total			8120	4341	69	7362	4413	52
Ratio			1.10	.98	1.33	1	1	1

We tested both ComMap and SeqMap on a set of circuits using 5-LUTs. The results are summarized in Table I. The test examples are derived from the multilevel sequential benchmark circuits in the LGSynth91 suite. The original benchmark circuits were decomposed into 2-bounded circuits using SIS command *tech_decomp* [Sentovich et al. 1992]. In Table I, under column **Initial**, we list the number of gates (excluding inverters) and the number of FFs in each decomposed circuit. Under column **ComMap**, we list the number of LUTs, the number of FFs, and the clock period (ϕ) of the mapping solution produced by ComMap. The same quantities are also listed for SeqMap. For SeqMap, we set the control parameter i to 6 in the experiments. (Thus, the clock periods of the mapping solutions produced by SeqMap may not be the minimum.) However, even with such a small depth, SeqMap produced mapping solutions with smaller clock periods for the circuits in the table consistently. This demonstrates the advantage of considering retiming and signal dependencies across FF boundaries. It can be seen also that the mapping solutions produced by SeqMap usually have fewer LUTs. This is also as expected, since SeqMap can form LUTs by extending across FF boundaries. Overall, the mapping solutions produced by ComMap have 10% more LUTs, 33% larger clock periods, and 2% fewer FFs. The CPU times of our current implementation of SeqMap were less than two minutes for all the test circuits except s38417, and in most cases they are only a few seconds on a SPARC 5 workstation with 32MB memory. However, it took SeqMap close to 30 minutes for s38417 due to the size of the circuit. Overall, the CPU times of SeqMap were about 10 times that of ComMap for the test circuits.

8. CONCLUSIONS

In this article, we proposed a novel approach to technology mapping for LUT-based FPGAs for sequential circuits. We studied the problem in a very general setting. In our approach, retiming is integrated into the mapping process. The approach is FF boundary–oblivious, as it implicitly considers all FF configurations that can be obtained by retiming. As a result, the issue of where to place the FFs (using retiming) in a circuit has no effect on our approach. On the other hand, for existing approaches, FF boundaries always exist, and signal dependencies across FF boundaries are severed. As a result, the solution space explored by our approach is much larger than that explored by existing approaches.

We also presented a polynomial mapping algorithm for our approach. The algorithm produces mapping solutions with optimal clock periods. Experimental results further demonstrated the superiority of the new approach. Currently, we are studying ways to further enhance and improve our algorithm. We are also investigating the possibility of applying the ideas we develop here for LUT reduction.

Besides a direct contribution to the technology mapping problem, this article introduced two important new concepts that have a more general applicability. The concept of expanded circuits is essentially a way to extract combinational logic in a sequential circuit across FF boundaries. It can be used to manipulate a sequential circuit using combinational techniques, while allowing dynamic FF positions. For example synthesis has been integrated recently into technology–mapping for sequential circuits using expanded circuits [Cong and Wu 1997]. The other important concept is l-values. l-values allow us to consider the effect of retiming in sequential synthesis and optimization.

APPENDIX

Proof of Theorem 3

Let $w_1(e)$ denote the number of FFs on edge e and $w_2(e)$ denote the l-weight of e. We use $w_1(p)$ and $w_2(p)$ to denote $\Sigma_{e \in p}\ w_1(e)$ and $\Sigma_{e \in p}\ w_2(e)$ for a path p in S, respectively, and $d(p)$ denote the number of LUTs (excluding PIs and POs) on p. Let $l(v)$ denote the l-value of node v in S.

(*Only if*). Suppose there is a path p from a PI u to a PO v such that $l(v) = w_2(p) > \phi$. That is, $w_2(p) = -\phi \cdot w_1(p) + d(p) > \phi$. Then, $d(p) > (w_1(p) + 1)\phi$. To have a clock period of ϕ on p, we have to have $\lceil d(p)/\phi \rceil - 1 \geq w_1(p) + 1$ FFs on p by pigeonhole principle. This is obviously impossible since retiming does not change the number of FFs on a path from a PI to a PO.

(*if*). Let r be the following retiming:

$$r(v) = \begin{cases} 0 & v \text{ is a PI or a PO} \\ \left\lceil \dfrac{l(v)}{\phi} \right\rceil - 1 & \text{otherwise.} \end{cases}$$

Let S_r denote the circuit obtained by retiming S according to r. We want to show S_r has a clock period of ϕ or less.

First, we verify that r is a legal retiming. That is, the number of FFs on each edge in S_r is non-negative. Let $u \xrightarrow{e} v$ be an edge in S. Then, $l(v) \geq l(u) + w_2(e) = l(u) - \phi \cdot w_1(e) + \delta(v)$. There are four cases depending on whether u and v are POs, PIs, or LUTs. All the case can be proved similarly. Here, we show the case that u is a LUT, and v is a PO. In this case, we have $\phi \geq l(v) \geq l(u) - \phi \cdot w_1(e)$. Dividing both sides by ϕ and taking the ceiling, we have $1 = r(v)(= 0) + 1 \geq \lceil l(u)/\phi \rceil - w_1(e) = r(u) + 1 - w_1(e)$. After rewriting, we have, $w_1(e) + r(v) - r(u) \geq 0$, i.e., e has non-negative weight in S_r.

To show that S_r has a clock period of ϕ or less, it suffices to show that for any path p such that $d(p) \geq \phi + 1$, there is at least one FF on p in S_r. Let the first and last nodes of p be u and v, respectively. Then,

$$l(v) \geq l(u) + w_2(p) = l(u) - \phi \cdot w_1(p) + d(p) - \delta(u).$$

Dividing both sides by ϕ, taking the ceiling, and subtracting 1, we have,

$$r(v) \geq r(u) - w_1(p) + \left\lceil \frac{d(p) - \delta(u)}{\phi} \right\rceil \geq r(u) - w_1(p) + 1,$$

so, $w_1(p) + r(v) - r(u) \geq 1$. Namely, there is at least one FF on p in S_r. □

ACKNOWLEDGMENTS

We thank the reviewers for their helpful suggestions, and Dr. Jason Cong and Mr. Chang Wu of UCLA for useful discussions. The first author would also like to thank Mr. Yuji Kukimoto of UC, Berkeley for several of his comments and suggestions that helped improve this paper.

REFERENCES

ALTERA. 1995. *Data Book*. Altera, San Jose, CA.

BHAT, N., AND HILL, D. 1992. Routable technology mapping for FPGAs. In *Proceedings of the ACM/SIGDA Workshop on FPGAs*, 143–148.

CONG, J., AND DING, Y. 1993. Beyond the combinational limit in depth minimization for LUT-based FPGA designs. In *Proceedings of the Digest IEEE/ACM International Conference on Computer-Aided Design* 110–114.

CONG, J., AND DING, Y. 1994. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. on Computer-Aided Design 13*, 1–11.

CONG, J., AND DING, Y. 1994. On area/depth trade-off in LUT-based FPGA technology mapping. *IEEE Trans. on VLSI Systems 2*, 137–148.

CONG, J., AND WU, C. 1996. An improved algorithm for performance optimal technology mapping with retiming in LUT-based FPGA design. In *Proceedings of the International Conference on Computer Design*, 572–578.

CONG, J., AND WU, C. 1997. Performance-driven FPGA synthesis with retiming and pipelining for sequential circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms.* McGraw-Hill Book Company, New York.

FARRAHI, A. H., AND SARRAFZADEH, M. 1994. Complexity of the lookup-table minimization problem for FPGA technology mapping. *IEEE Trans. on Computer-Aided Design 13*, 1319–1332.

FRANCIS, R. J., ROSE, J., AND CHUNG, K. 1990. Chortle: A technology mapping for lookup table-based field programmable gate arrays. In *Proceedings of the ACM/IEEE Design Automation Conference*, 613–619.

FRANCIS, R. J., ROSE, J., AND VRANESIC, Z. 1991. Chortle-crf: Fast technology mapping for lookup table-based FPGAs. In *Proceedings of the ACM/IEEE Design Automation Conference*, 227–233.

FRANCIS, R. J., ROSE, J., AND VRANESIC, Z. 1991. Technology mapping for lookup table-based FPGAs for performance. In *Digest of the IEEE/ACM International Conference on Computer-Aided Design*, 568–571.

KARPLUS, K. 1991. Xmap: A technology mapper for table-lookup FPGAs. In *Proceedings of the ACM/IEEE Design Automation Conference*, 240–243.

LEISERSON, C. E., AND SAXE, J. B. 1991. Retiming synchronous circuitry. *Algorithmica 6*, 5–35.

MATHUR, A., AND LIU, C. L. 1994. Performance driven technology mapping for lookup-table based FPGAs using the general delay model. In *Proceedings of the ACM/SIGDA Workshop on Field Programmable Gate Arrays*.

AT&T MICROELECTRONICS. 1995. *AT&T Field-Programmable Gate Arrays Data Book*. AT&T Microelectronics.

MURGAI, R., BRAYTON, R., AND SANGIOVANNI-VINCENTELLI, A. 1993. Sequential synthesis for table look up programmable gate arrays. In *Proceedings of the ACM/IEEE Design Automation Conference*, 224–229.

MURGAI, R., NISHIZAKI, Y., SHENOY, N., BRAYTON, R., AND SANGIOVANNI-VINCENTELLI, A. 1990. Logic synthesis algorithms for table look up programmable gate arrays. In *Proceedings of the ACM/IEEE Design Automation Conference*, 620–625.

MURGAI, R., SHENOY, N., BRAYTON, R., AND SANGIOVANNI-VINCENTELLI, A. 1991. Improved logic synthesis algorithms for table look up architectures. In *Digest of the IEEE/ACM International Conference on Computer-Aided Design*, 564–567.

PAN, P. To appear. Continuous retiming: algorithms and applications. In *International Conference on Computer Design (ICCD)*.

PAN, P., AND LIU, C. L. 1996. Optimal clock period FPGA technology mapping for sequential circuits. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 720–725.

PAN, P., AND LIU, C. L. 1996. Technology mapping of sequential circuits for LUT-based FPGAs for performance. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 58–64.

SAWKAR, P., AND THOMAS, D. 1992. Area and delay mapping for table-look-up based field programmable gate arrays. In *Proceedings of the ACM/IEEE Design Automation Conference*, 368–373.

SAWKAR, P., AND THOMAS, D. 1993. Performance directed technology mapping for look-up table based FPGAs. In *Proceedings of the ACM/IEEE Design Automation Conference*, 208–212.

SCHLAG, M., KONG, J., AND CHAN, P. 1994. Routability-driven technology mapping for lookup table-based FPGA's. *IEEE Trans. on Computer-Aided Design 13*, 13–26.

SENTOVICH, E. M., ET AL. 1992. Sequential circuit design using synthesis and optimization. In *Proceedings of the International Conference on Computer Design*, 328–333.

WEINMANN, U., AND ROSENSTIEL, W. 1993. Technology mapping for sequential circuits based on retiming techniques. In *Proceedings of the European Design Automation Conference*, 318–323.

WOO, N.-S. 1991. A heuristic method for FPGA technology mapping based on the edge visibility. In *Proceedings of the ACM/IEEE Design Automation Conference*, 248–251.

XILINX. 1993. *The Programmable Gate Arrays Data Book.* Xilinx, San Jose, CA.

YANG, H., AND WONG, D. F. 1994. Edge-Map: Optimal performance driven technology mapping for iterative LUT based FPGA designs. In *Digest of the IEEE/ACM International Conference on Computer-Aided Design*, 150–155.

Received February 1996; revised October 1996; accepted August 1997