A New Retiming-based Technology Mapping Algorithm for LUT-based FPGAs

Peichen Pan[†] and Chih-Chang Lin[‡]
[†]Dept. of ECE, Clarkson University, Potsdam, NY 13699
[‡]Verplex Systems, Inc., San Jose, CA 95112

Abstract

In this paper, we present a new retiming-based technology mapping algorithm for look-up table-based field programmable gate arrays. The algorithm is based on a novel iterative procedure for computing all k-cuts of all nodes in a sequential circuit, in the presence of retiming. The algorithm completely avoids flow computation which is the bottleneck of previous algorithms. Due to the fact that k is very small in practice, the procedure for computing all k-cuts is very fast. Experimental results indicate the overall algorithm is very efficient in practice.

1 Introduction

A look-up table-based FPGA consists of an array of programmable logic blocks together with programmable interconnects [1, 27, 15]. The core of a programmable logic block is a k-input look-up table (k-LUT) which can implement any combinational logic with up to k inputs and a single output, where k is a small positive integer.

The technology mapping problem for LUT-based FPGAs is to produce an equivalent circuit comprised of k-LUTs for a given circuit. The problem has been extensively studied, but most research focused on combinational circuits. Mapping algorithms for combinational circuits have been proposed for performance [3, 11, 14, 23, 28], area [8, 9, 10, 12, 17, 26], routability [2, 24], and combinations of these [4, 22]. In particular, delay-optimal mapping algorithms have be proposed for combinational circuits [3, 28].

There were only a few mapping algorithms targeted for sequential circuits [16, 25, 20]. The standard approach is to simply cut the circuits by removing all FFs, then map the remaining combinational logic. This approach ignores signal dependencies across FF boundaries and the possibility of exposing the combinational logic between FFs in different ways, both of which are made available by retiming [13].

Recently, a new approach to the technology mapping problem for LUT-based FPGAs was proposed [19, 21]. In this approach there is no circuit cutting to remove FFs. Moreover, FF positions are assumed to be dynamic in that they can be repositioned by retiming. As a result, signal dependencies across FF boundaries are exploited. The authors proposed a polynomial mapping algorithm that can obtain a mapping solution with minimum clock period. The algorithm is based on two important concepts: *l-values and expanded circuits.* The expanded circuit for a node captures the information on all the k-LUTs at the node, while taking into consideration both retiming and logic replication. l-values can be used to determine critical sequential paths.

Although it is polynomial, the algorithm proposed in [19, 21] can be slow in practice due to repeated network flow computation on graphs of large size. The amount of flow computation is enormous for a circuit of moderate size since, in general, several passes of flow computation need to be carried out at all nodes in a circuit for a given target clock period. Moreover, if a minimum clock period mapping solution is desired, several target clock periods may need to be examined. In fact, the running time of the algorithm is so high that a controlling parameter has to be introduced to trade-off solution quality for reduced running time. Recently, several techniques have been proposed to speed up this algorithm [6, 7]. These techniques reduce both the size of the graphs used in flow computation and and the number of times that flow computation is carried out at each node. Significant improvement in running time was observed, but the amount of flow computation is still very large.

In this paper, we present a new performance-driven technology mapping algorithm for sequential circuits. The algorithm is based on a novel iterative procedure that computes all k-cuts of all nodes in a sequential circuit. The procedure is very efficient in practice due to the fact that k is very small. With all k-cuts available, the rest of the algorithm is very efficient since network flow computation is completely avoided, and in its place are some simple arithmetic operations. The saving is even greater if an optimal clock period mapping solution is sought since the k-cuts need only be computed once even though several target clock periods may need to be examined. The overall algorithm is very efficient in practice. Experimental results show the algorithm can handle the largest ISCAS circuits with ease.

Since all k-cuts are available, our approach has potential to consider other criteria such as equivalent initial states and LUT minimization, in addition to performance. In fact our algorithm employs a heuristic to minimize the number of LUTs.

The remainder of the paper is organized as follows: Section 2 presents some definitions and concepts. In Section 3, we present the procedure for generating all k-cuts of all nodes in a sequential circuit. The overall technology mapping algorithm is described in Section 4. We present some experimental results in Section 5. Finally, Section 6 concludes this paper.

2 Preliminaries

A (sequential) circuit can be modeled as an edge-weighted directed graph. The nodes are the primary inputs (PIs), the primary outputs (POs), and the combinational elements (e.g., gates and k-LUTs). The edges represent the interconnections. There is an edge e from u to v (denoted $u \xrightarrow{e} v$) with weight t if the output of u, after passing through t FFs, is an input to v. We will use N to denote the circuit to be mapped and $w(e)$ to denote the weight of an edge e in N. We also assume that every node in N can be reached from at least one PI and can reach at least one PO.

The *clock period* of a circuit is the maximum delay on the combinational paths (paths without FFs) in the circuit. In this paper, we use the unit-delay model when we calculate the clock period of a mapping solution. That is, each LUT has one unit of delay and interconnection has no delay.

Retiming is a technique of repositioning the FFs in a circuit without changing the functionality or the structure of the circuit [13]. A retiming r can be represented as a function from the nodes to integers where $r(v)$ denotes the retiming value at node v. In the circuit retimed according to r, the weight of an edge $u \xrightarrow{e} v$ becomes $w(e) + r(v) - r(u)$.

As has been demonstrated [19], we can obtain better mapping solutions that are otherwise impossible, by integrating retiming into technology mapping. In this paper, we study the technology mapping problem in conjunction with retiming. The objective is to find a mapping solution with minimum clock period. As in [19], we consider the decision version of the problem: *Given a target clock period ϕ, find a mapping solution with a clock period of ϕ or less, whenever such a mapping solution exists.* With an algorithm for the decision problem, we can carry out binary search on the target clock period to find a mapping solution with minimum clock period, if such a solution is desired. Note that the minimum clock period is obviously between 1 and n, where n is the number of nodes in N.

One important concept in technology mapping for sequential circuits is l-values [19]. Consider a mapping solution S. For each edge $u \xrightarrow{e} v$ in S, we assign an l-weight, $-\phi \cdot d + \delta(v)$, to it, where d is the number of FFs on e, and $\delta(v) = 1$ if v is a LUT or 0 if it is a PI or PO. *The l-value of a node in S is defined as the maximum weight of the paths from the PIs to the node using the l-weights.* l-values have the following property [19, 21]:

Theorem 1 *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.*

It is evident from the above result that the concept of l-values[1] is very similar to arrival times used in combinational synthesis and optimization. In fact, if a circuit is combinational, l-values reduce naturally to arrival times. As a result of Theorem 1, the technology mapping problem is then reduced to that of *finding a mapping solution with minimum l-values at the POs.*

To find the minimum l-value, a method is needed to examine all k-LUTs at each node. The concept of expanded circuits was introduced for this purpose [19]. The expanded circuit for a node v is formed by properly replicating the nodes in N starting from v and going backward towards PIs. It is constructed in such a way that all paths from any node to the only output node have the same number of FFs. In the expanded circuit for v, each node is a copy of a node, say u, in N and is denoted by u^d where the index d is the number of FFs on a path from the copy to the only output node which is v^0.

Expanded circuits are constructed recursively. To construct the expanded circuit for a node v in N, we start with v^0, then repeatedly carry out *expansion* at nodes that do not have incoming edges. Let u^d be one such node. An *expansion* at u^d refers to the operation that for each edge $x \xrightarrow{e} u$ in N, add node x^{d_1}, where $d_1 = d + w(e)$, to the expanded circuit if it is not there, and add an edge $x^{d_1} \rightarrow u^d$ with weight $w(e)$ to the expanded circuit. For the circuit in Fig. 1(1), Fig. 1 shows three expansions in the construction of the expanded circuit for g. From (2) to (5) each one is obtained from the preceding one by expanding at the shaded node.

The expanded circuit for v is a DAG with one sink v^0. Due to possible presence of cycles, the expanded circuit may have infinite many nodes. This is obviously the case for node g in Fig. 1(1), since expansion will be continued at g^1 in Fig. 1(5). Note that the expanded circuit becomes repetitive in this case.

In the expanded circuit for v, a *cut* $(X, \overline{X})$ is a partition of the nodes such that v^0 is in X and all sources are in $\overline{X}$. The *node-set* of the cut is the set of nodes in $\overline{X}$ that are connected to one or more nodes in X. The *cone* of the cut is the subgraph induced by X. If the size of the node-set of a cut is less than or equal to k, the cut is further called a *k-cut*. The following result was presented in [19, 21]:

Theorem 2 *Given a k-cut in the expanded circuit for v, a k-LUT at v can be derived. The logic of the LUT is the cone of the cut less the FFs. u, after passing d FFs, is an input to the LUT if u^d is in the node-set of the cut. Moreover, for any k-LUT at v there exists a k-cut in the expanded circuit for v that derives a k-LUT with the same set of inputs.*

The dotted line in the expanded circuit in Fig. 1(5) indicates a 3-cut as the node-set consists of three nodes i_1^0, g^1 and i_2^1. The corresponding 3-LUT is shown in Fig. 2.

3 A procedure for generating all k-cuts

To determine the minimum l-values of the POs we need a way to examine all k-LUTs at each node. We also need

[1] l-values are a special version of the so-called *continuous retiming* introduced in [18].

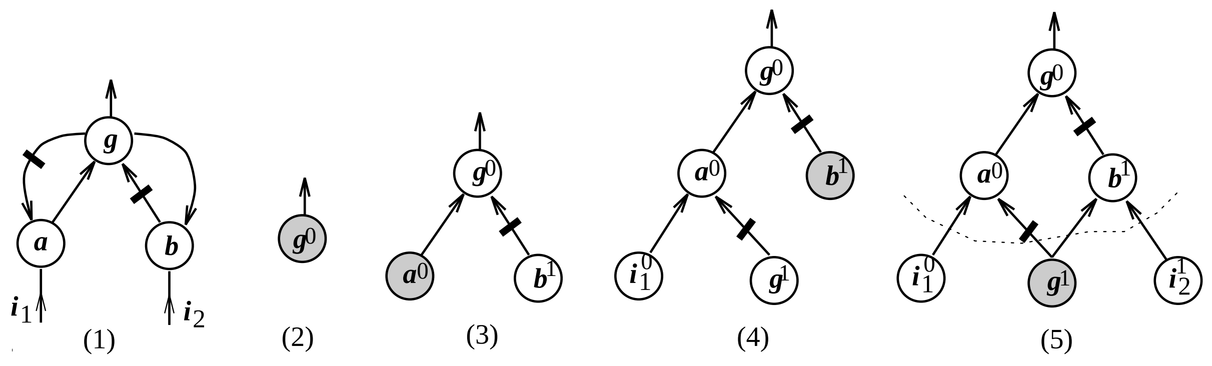

Figure 1: Construction of expanded circuits.

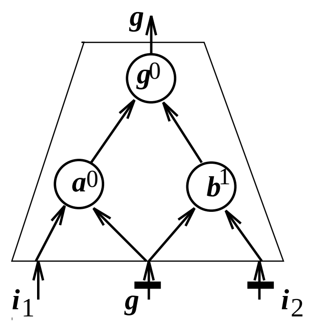

Figure 2: A LUT derived from a cut.

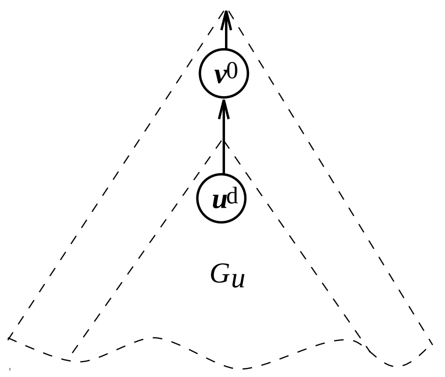

Figure 3: Iterative structure of an expanded circuit.

k-LUTs to construct the final mapping solution. Because of the correspondence of k-LUTs and k-cuts as stated in Theorem 2, it suffices to find all k-cuts of each node. In this section, we present a procedure for computing all k-cuts of all nodes in N. The procedure actually computes the node-sets of all k-cuts. Later, we will discuss how to obtain the cones so that we can determine the logic of the corresponding k-LUTs.

The node-sets of cuts can also be characterized independent of the underlying cuts. *A set of nodes form a node-set of a cut if by removing all nodes in the set, we can break all paths from the sources to the sink.* In fact, all nodes that can still reach the sink after the removal form one part of a cut and all other nodes (including those in the set) form the other part of the cut. As a slight abuse of notation, we will use the term cut loosely to also mean a set of nodes that can be the node-set of a cut. As a convention, we also view $\{v^0\}$ as a cut of v and refer it as the *trivial cut*. Note that trivial cuts fit the above new characterization of cuts. Trivial cuts are important for determining other k-cuts although they cannot be used in a mapping solution, as they are not the node-set of any cut.

To understand our procedure for computing all k-cuts, it is useful to examine the structure of an expanded circuit. Suppose $u \overset{e}{\to} v$ is an edge in N and $w(e) = d$. Then, $u^d \to v^0$ is an edge in the expanded circuit for v. It is evident from the definition of expanded circuits that further expansion down from u^d is not different from the construction of the expanded circuit for u, except for the latter, expansion starts with u^0. Let G_u denote the subgraph rooted at u^d in the expanded circuit for v, as indicated in Fig. 3. If we subtract d from the index of each node in G_u, the resulting graph is then the expanded circuit for u. In general, for any node in an expanded circuit its transitive fan-ins together with the node form the expanded circuit for the node if we subtract the index of the node from the indices of all nodes.

Let u_1, u_2, ..., u_t be the fan-ins of a node v in N. Let d_i denote the number of FFs on the edge from u_i to v, and C_{u_i} denote the set of k-cuts of u_i for $1 \le i \le t$. We define a set operation called *merge* as follows:

$$merge(C_{u_1}, C_{u_2}, \cdots, C_{u_t}) =$$
$$\{c = c_1^{d_1} \cup c_2^{d_2} \cup \cdots \cup c_t^{d_t} \mid c_i \in C_{u_i} \text{ and } |c| \le k\},$$

where $c_i^{d_i} = \{x^{d+d_i} \mid x^d \in c_i\}$.

The following result is the basis of the procedure for generating all k-cuts. (The proofs of all results are omitted due to space limitation.)

Theorem 3 *The set of k-cuts of v, C_v is equal to*

$$merge(C_{u_1}, C_{u_2}, \cdots, C_{u_t}) \cup \{\{v^0\}\}.$$

From Theorem 3, if the k-cuts of the fan-ins of a node are given, we can find all non-trivial k-cuts of the node using the *merge* operation. For a circuit without feedback loops, we can compute the k-cuts of all nodes by examining the nodes in topological order starting from the PIs. At each node, the *merge* operation is applied to obtain all its k-cuts. Note that a PI only has the trivial cut. In practice, most circuits contain loops. For such circuits, this idea cannot be directly applied since the k-cuts of the nodes are cyclically dependent on each other. For example, in the circuit in Fig. 1(1), to obtain all k-cuts of node g using the *merge* operation, we need to have all k-cuts of node a, and vice versa.

For circuits with loops, we still can use Theorem 3 to determine k-cuts although one pass of merging may not be enough. Our approach is to determine all k-cuts of all nodes by successive approximation, again using the *merge* operation. For each node v in N, we maintain a subset L_v of C_v and successively update L_v by adding more and more k-cuts. We update L_v by merging the current subsets for the fan-ins of v. This process is repeated until no further addition is possible for any of the subsets. Initially, for each non-PO node v, the subset contains the only known k-cut — its trivial k-cut $\{v^0\}$. Since a PO is not mapped, a PO v has only one k-cut $\{u^d\}$, where u is the node that drives v and d is the number of FFs on the edge. The procedure is summarized in Fig. 4.

```
FIND_ALL_CUTS($N, k$)
1.  for each non-PO node $v$ in $N$ do
2.      $L_v = \{\{v^0\}\}$;
3.  Done = FALSE;
4.  while (Done == FALSE) do
5.      Done = TRUE;
6.      for each node $v$ (not PI or PO) in $N$ do
7.          $tmp = merge(L_{u_1}, L_{u_2}, \cdots, L_{u_t})$;
8.          if tmp $\not\subseteq L_v$ then
9.              $L_v = tmp \cup \{\{v^0\}\}$;
10.             Done = FALSE;
11. return success    // $L_v$ has settled to $C_v$ for each $v$.
```

Figure 4: Procedure for generating all k-cuts.

We now use an example to illustrate the procedure. Consider the circuit in Fig. 1(1) with $k = 3$. Initially, $L_x = \{\{x^0\}\}$ for each non-PO node x. Suppose we examine the nodes a, b, and g in this order and carry out the *merge* operation. Consider the first iteration. For gate a,

$$\begin{aligned} tmp &= merge(L_{i_1}, L_g) \\ &= \{c_1^0 \cup c_2^1 \mid c_1 \in L_{i_1}, c_2 \in L_g, |c_1^0 \cup c_2^1| \le 3\} \\ &= \{\{i_1^0, g^1\}\}, \end{aligned}$$

so, after the updating in line 9, $L_a = \{\{a^0\}, \{i_1^0, g^1\}\}$. Similarly, for gate b after the updating, $L_b = \{\{b^0\}, \{i_2^0, g^0\}\}$. Now for gate g,

$$\begin{aligned} tmp &= merge(L_a, L_b) \\ &= \{c_1^0 \cup c_2^1 \mid c_1 \in L_a, c_2 \in L_b, |c_1^0 \cup c_2^1| \le 3\} \\ &= \{\{a^0, b^1\}, \{a^0, i_2^1, g^1\}, \{i_1^0, g^1, b^1\}, \{i_1^0, g^1, i_2^1\}\}. \end{aligned}$$

Table 1 lists the cuts and the iteration in which they are generated. After two iterations, all subsets stabilized and all 3-cuts of all nodes are found. Note that for the PO o, $C_o = \{\{g^0\}\}$.

itr	i_1	i_2	a	b	g
0	$\{i_1^0\}$	$\{i_2^0\}$	$\{a^0\}$	$\{b^0\}$	$\{g^0\}$
1			$\{i_1^0, g^1\}$	$\{i_2^0, g^0\}$	$\{a^0, b^1\}$ $\{a^0, i_2^1, g^1\}$ $\{i_1^0, g^1, b^1\}$ $\{i_1^0, g^1, i_2^1\}$
2			$\{i_1^0, a^1, b^2\}$	$\{i_2^0, a^0, b^1\}$	

Table 1: Generating all cuts, an example.

Lemma 1 *At line 7 in the procedure* FIND_ALL_CUTS*,* $L_v \subseteq tmp \cup \{\{v^0\}\}$.

From the above lemma, we know that after each *merge* operation the updated subset contains the subset before the *merge* operation. Each *merge* may also discover some new k-cuts. The subset for each node becomes larger and larger (at least keeps the same), as more and more *merge* operations are carried out.

Another implication of Lemma 1 is that we can replace the test in line 8 in the procedure by $|tmp| > |L_v| - 1$ (note that L_v contains the trivial cut, but tmp does not) and avoid the more expensive set operation of testing $tmp \not\subseteq L_v$.

We now show any k-cut of any node in N will be discovered sooner or later. Let c be a k-cut of v. From Theorem 2, we know c is a k-cut in the expanded circuit for v. Let $p(c)$ denote the number of edges on a path with the largest number of edges from the nodes in the cut to the sink v^0. We have the following result:

Lemma 2 *After m iterations of the* **while** *loop in the procedure* FIND_ALL_CUTS*,*

$$L_v \supseteq \{c \mid c \textit{ is a } k\textit{-cut of } v \textit{ such that } p(c) \le m\}.$$

It has been shown that $p(c) \le kn$ for any k-cut c of any node, where n is the number of nodes in N [19]. Therefore, the procedure will find all k-cuts of all nodes after at most kn iterations. Of course, this is the worst-case scenario. One nice feature of the procedure is that if the maximum $p(c)$ for all cuts is D, the number of iterations is at most D. In practice, we expect D to be substantially smaller than kn.

To further reduce the number of iterations, we arrange the nodes in N in a particular order and call the *merge* operation in that order during each iteration. The guiding principle in choosing an order is to carry out the *merge* operation at a node after at its fan-ins as much as possible. The particular order we use is obtained by removing all outgoing edges of a feedback vertex set in N and topologically order all nodes starting from those nodes that do not have incoming edges. Using this order, the procedure stopped in at most five iterations for all the ISCAS89 benchmark circuits when $k = 4$.

The basic procedure can be further improved in several aspects. An obvious one is to add an event-driven mechanism to the procedure so that we do not carry out the *merge* operation on a node if none of its fan-ins have new cuts added in the previous iteration.

From Lemma 1, it is obvious that in addition to some possible new ones, the *merge* operation will regenerate the cuts that are already in L_v. To remove the redundant work, we number the *merge* operations sequentially and time-stamp the cuts by the *merge* operation in which they are generated. If a cut c is introduced in the q-th *merge* operation, c gets a time-stamp $s(c) = q$. We also associate each subset L_v with a time-stamp $s(L_v)$, representing the latest *merge* operation carried out at v. We then modify the *merge* operation so that it only combines those cuts involving at least one cut with a time-stamp larger than the time-stamp of L_v. That is,

$$\begin{aligned}&merge(L_1, L_2, \cdots, L_t) = \\ &\{c = c_1^{d_1} \cup c_2^{d_2} \cup \cdots \cup c_t^{d_t} \mid \\ &c_i \in C_{u_i}, \max_i s(c_i) > s(L_v), |c| \leq k\}.\end{aligned}$$

If a cut in the above set is indeed new (not in current L_v), we stamp it by the current *merge* operation and insert it into L_v. To carry out the insertion efficiently, we organize L_v as a hash table. With a hash table, the insertion can be down in an expected time $O(1)$, independent of the size of L_v.

Finally, we discuss how to obtain k-LUTs from the k-cuts. Remember that what FIND_ALL_CUTS computes are actually the node-sets of the cuts. In other words, they are the input sets of the k-LUTs. Suppose c is a k-cut of v. To generate the corresponding k-LUT, we start with node v^0 and carry out expansion repeatedly just as we construct the expanded circuit for v, until all sources fall into c. For example, for the 3-cut $\{i_1^0, g^1, i_2^1\}$ of g in Fig. 1(5), the expansion leads to the expanded circuit in Fig. 1(5), and the corresponding k-LUT is exact the one shown in Fig. 2.

4 The technology mapping algorithm

We now describe the technology mapping algorithm assuming all k-cuts of all nodes in N have been determined. The algorithm first finds the minimum l-values of the POs. It then selects k-cuts to form a mapping solution that meets the minimum l-values while trying to minimize the number of LUTs. Finally, for each node (signal) remained in the mapping solution, the k-LUT is derived from the k-cut selected for the node. This last step is discussed in the previous section. In the rest of this section, we discuss the details of the first two steps.

4.1 Finding the minimum l-values

We determine the minimum l-values of the POs by finding the minimum l-values of all nodes in N, using dynamic programming. The approach is to maintain a lower bound $l(v)$ on the minimum l-value of each node v and successively improve the bound until no further improvement is possible, which indicates that the lower bounds have settled to the minimum l-values.

The l-values of the PIs are always zero. Initially, the lower-bounds for all non-PI nodes are set to $-\infty$. Suppose c is a k-cut of v and u^d is in c. If the k-LUT derived from c is selected for v, then u must be in the mapping solution and there is an edge from u to v with d FFs according to Theorem 2. Therefore, based on the current lower bound, the l-value at v will be at least $\max\{l(u) - \phi \cdot d + \delta(v) \mid u^d \in c\}$. To minimize the l-value of v, we minimize the new bound for v by updating $l(v)$ according to the following formula:

$$l(v) = \min_{c, \text{ a } k\text{-cut of } v} \left(\max\{l(u) - \phi \cdot d + \delta(v) \mid u^d \in c\}\right).$$

In previous algorithms [19, 6], the new bound is determined by solving a max-flow problem on a network derived from the expanded circuit for v. In our case, since C_v, the set of k-cuts of v, has been computed, the new lower bound for v can be determined by simply examining all k-cuts in C_v. For each non-trivial k-cut c in C_v, we calculate $\max\{l(u) - \phi \cdot d + \delta(v) \mid u^d \in c\}$ and update $l(v)$ to the minimum of all such values. Of course, if a lower bound for a PO is larger than ϕ at any moment, we can stop the algorithm as it becomes obvious that there is no mapping solution with a clock period of ϕ. Fig. 5 summarizes the procedure.

```
FIND_MIN_VALUES($N, \phi$)
1. **for** each node $v$ in $N$ **do**
2.   **if** $v$ is a PI **then** $l(v) \leftarrow 0$;
     **else** $l(v) \leftarrow -\infty$;
3. Done $\leftarrow$ FALSE;
4. **while** Done == FALSE **do**
5.   Done $\leftarrow$ TRUE;
6.   **for** each non-PI node $v$ in $N$ **do**
7.     $tmp = \min_{c \in C_v - \{v^0\}}(\max\{l(u) - \phi \cdot d + \delta(v) \mid u^d \in c\})$
8.     **if** $tmp > l(v)$ **then**
9.         $l(v) = tmp$;
10.        Done = FALSE;
11.    **if** $v$ is a PO and $l(v) > \phi$, **return** failure;
12**return** success;    // Bounds have settled.
```

Figure 5: Procedure for computing the the minimum l-values.

4.2 Constructing a mapping solution

In this step, we generate a mapping solution with a clock period of ϕ. Remember that for each node v in N, its minimum l-value (denoted $l^{opt}(v)$) has already been determined in the previous step and the minimum l-values of the POs are all less than or equal to ϕ.

We start the construction at the POs and proceed backward to select nodes and k-cuts to form a mapping solution. More specifically, we maintain a set U consisting of the nodes that will be included in the final mapping solution, but their k-cuts are yet to be selected. Initially, the mapping solution S consists of the PIs and POs, and U consists of only the POs. At each step, a node v is removed from U. A k-cut c

of v is selected to include in S (details later). For each u^d in c, we add u to both U and S if it is not already in S. An edge with d FFs from u to v is then created in S. This process continues until U becomes empty.

We now discuss how to select a k-cut for a node v removed from U. To guarantee the minimum l-values at POs, we require the cut c achieve the minimum l-value at v. Namely, $l^{opt}(v) = \max\{l^{opt}(u) - \phi \cdot d + \delta(v) \mid u^d \in c\}$. (This requirement is not necessary though. In fact, relaxing this requirement may further reduce the number of LUTs.) In general, there are several such k-cuts. We want to choose one that could potentially reduce the number of LUTs in the final mapping solution. Intuitively, a node with a large number of fan-outs in N is very likely to appear in a mapping solution. Consequently, we assign a cost to each cut c, $cost(c) = \sum \frac{1}{fanout(u)}$, where $fanout(u)$ is the number of fan-outs at v in N. The summation is taken over all u^d in c such that u is not currently in S. If $cost(c)$ is small, the nodes in c are either already in S or have a large number of fan-outs, so c is a good candidate k-cut to select for v. Thus, we select a cut for v that has the minimum $cost$. The procedure is described in Fig. 6.

```
CONSTRUCT_SOLUTION($N, \phi, l^{opt}$)
1.  $U \leftarrow$ the set of POs;
2.  $S \leftarrow \{v \mid v$ is a PI or PI$\}$;
3.  while $U \neq \emptyset$ do
4.      $v \leftarrow$ a node in $U$;
5.      $U \leftarrow U - \{v\}$;
5.      $cost_{min} = \infty$;
6.      for each non-trivial cut $c \in C_v$ do
7.          if ($l^{opt}(v) == \max\{l^{opt}(u) - \phi \cdot d + \delta(v) \mid u^d \in c\}$
                and $cost_{min} > cost(c)$) then
8.              $cost_{min} = cost(c)$; $c_{best} = c$;
9.      for each $u^d \in c_{best}$ do
10.         if $u$ is not in $S$ then
11.             $S \leftarrow S \cup \{u\}$;
12.             $U \leftarrow U \cup \{u\}$;
13.         create an edge in $S$ from $u$ to $v$ with $d$ FFs;
14. return $S$;
```

Figure 6: Procedure for constructing a mapping solution.

Finally, we apply the following retiming to S to obtain a mapping solution with a clock period of ϕ or less:

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

5 Experimental results

We implemented a prototype technology mapping package called SeqMap-cut which searches the target clock period to find a mapping solution with minimum clock period. For a given clock period, SeqMap-cut employs the algorithm presented in this paper to find a mapping solution with the clock period. Experiments were carried out on a set of ISCAS89 circuits. In this section, we describe our experiments and summarize the results.

For each benchmark circuit, we first performed technology independent optimization, then decomposed all gates into two-input simple gates to arrive at the circuit used in our experiment. The sizes of the circuits range from less than a hundred to over ten thousand gates (excluding inverters).

In the experiment we set k to 4 to reflect commercial FPGA architectures [1, 27]. The experimental results are summarized in Table 2. Column **SeqMap-cut** lists the number of LUTs, number of FFs and the clock period of each mapping solution from SeqMap-cut. Note that the clock period of the mapping solution from SeqMap-cut is optimal. For comparison purpose we also list the statistics of the mapping solutions from a mapping program called ComMap-cut. ComMap-cut first maps the combinational logic between the FFs optimally, using the algorithm for a target clock period proposed in this paper[2]. (Note that when the algorithm is applied to a combinational circuit, it produces a mapping solution with minimum level as FlowMap.) It then retimes the mapping solution to its optimal clock period. For all examples in the table, SeqMap-cut produced better solutions in terms of clock period. It is also evident from the table that a mapping approach based on mapping combinational circuits between FFs may miss minimum clock period mapping solutions, which are guaranteed by SeqMap-cut.

In Table 2, we also list the CPU time of SeqMap-cut on a SPARC5 with 32MB memory. Under **gencut**, we list the CPU time of the procedure for generating all cuts. The total CPU time for finding the minimum l-values for all target clock periods is listed under **label**. We also list the number of passes of merging (the number of iterations of the *while* loop) in FIND_ALL_CUTS. It is obvious from the table that SeqMap-cut is very efficient and can handle large designs.

6 Conclusions

In this paper, we proposed a new technology mapping algorithm for LUT-based FPGAs. The algorithm is based on a novel iterative procedure to compute all k-cuts of all nodes in a sequential circuit. It can find a mapping solution with minimum clock period while minimizing the number of LUTs. Experimental results show the algorithm is very efficient in practice.

Further research is needed in the direction of minimizing the number of LUTs. A retimed circuit may not have an equivalent initial state. If the initial state is an integral part of a design, care must be taken in selecting cuts to form a mapping solution so that an equivalent initial state can be found. We believe our approach has potential to consider these and other issues, since all k-cuts of all nodes are available.

References

[1] Altera. *Data Book*. Altera, San Jose, CA, 1995.

[2] N. Bhat and D. Hill. Routable technology mapping for

[2]It was brought to our attention by Dr. Cong that the special case of computing cuts in combinational circuits was also proposed by other researchers [5].

test circuit			**ComMap-cut**			**SeqMap-cut**			**CPU time (s)**		no.
name	gates	FFs	LUTs	FFs	ϕ	LUTs	FFs	ϕ	gencut	label	itr
s208	70	8	28	8	4	30	14	3	0.84	0.04	5
s298	125	14	39	14	3	46	32	2	0.23	0.04	4
s344	109	15	41	15	4	36	26	3	0.34	0.05	3
s349	112	15	41	15	4	37	26	3	0.38	0.06	3
s382	144	21	61	21	5	70	38	4	0.34	0.11	4
s386	115	6	64	6	6	67	11	5	0.19	0.07	4
s400	139	21	58	21	6	61	31	4	0.33	0.17	4
s420	165	16	68	16	5	65	27	4	0.84	0.05	4
s444	138	21	61	21	6	66	34	4	0.33	0.11	4
s499	187	22	90	22	6	81	44	5	0.33	0.15	3
s510	213	6	111	6	5	117	28	4	0.42	0.05	3
s526	169	21	69	21	4	84	38	3	0.37	0.06	4
s635	201	32	68	47	6	77	50	5	0.61	0.05	3
s820	468	5	168	5	5	174	11	4	0.52	0.16	3
s832	303	5	164	5	6	165	8	5	0.39	0.11	3
s953	402	29	218	29	5	217	43	4	0.57	0.18	3
s967	369	29	200	29	5	200	55	4	0.49	0.14	3
s1269	489	37	195	53	8	204	76	6	1.72	0.30	4
s1488	734	6	314	6	6	325	8	5	1.14	0.28	3
s1494	746	6	320	6	6	338	8	5	1.13	0.26	3
s1512	642	57	268	57	11	293	82	9	1.51	0.74	3
s3271	1177	116	529	292	5	554	292	4	3.05	0.29	3
s3330	689	65	345	157	5	383	175	4	0.92	0.14	3
s4863	1458	83	398	150	8	393	142	7	10.01	0.52	3
s5378	1303	163	549	305	7	582	297	6	5.29	0.31	5
s6669	2263	231	555	339	7	607	364	6	13.63	0.71	3
s9234	943	138	386	207	6	389	217	5	3.27	0.44	4
s13207	2244	484	878	478	10	932	921	9	6.73	0.81	4
s15850	3238	504	1315	642	12	1390	701	11	9.03	1.31	4
s35932	12204	1728	3424	1728	4	2651	2034	3	84.44	7.06	3
s38417	9849	1583	3355	2182	9	3620	2394	8	74.65	29.34	3
s38584	13503	1426	4976	2678	8	4815	2651	7	47.13	6.65	4

Table 2: Experimental results.

FPGAs. In *ACM/SIGDA Workshop on FPGAs*, pages 143–148, 1992.

[3] J. Cong and Y. Ding. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. on Computer-Aided Design*, 13:1–11, 1994.

[4] J. Cong and Y. Ding. On area/depth trade-off in LUT-based FPGA technology mapping. *IEEE Trans. on VLSI Systems*, 2:137–148, 1994.

[5] J. Cong and Y.-Y. Hwang. Partially-dependent functional decomposition with applications in FPGA synthesis and mapping. In *ACM/SIGDA Symp. on Field Programmable Gate Arrays*, 1997.

[6] J. Cong and C. Wu. An improved algorithm for performance optimal technology mapping with retiming in LUT-based FPGA design. In *Proc. Intl. Conf. on Computer Design*, pages 572–578, 1996.

[7] J. Cong and C. Wu. Performance-driven FPGA synthesis with retiming and pipelining for sequential circuits. In *Proc. ACM/IEEE Design Automation Conf.*, 1997.

[8] A. H. Farrahi and M. Sarrafzadeh. Complexity of the lookup-table minimization problem for FPGA technology mapping. *IEEE Trans. on Computer-Aided Design*, 13:1319–1332, 1994.

[9] R. J. Francis, J. Rose, and K. Chung. Chortle: A technology mapping for lookup table-based field programmable gate arrays. In *Proc. ACM/IEEE Design Automation Conf.*, pages 613–619, 1990.

[10] R. J. Francis, J. Rose, and Z. Vranesic. Chortle-crf: Fast technology mapping for lookup table-based FPGAs. In *Proc. ACM/IEEE Design Automation Conf.*, pages 227–233, 1991.

[11] R. J. Francis, J. Rose, and Z. Vranesic. Technology mapping for lookup table-based FPGAs for performance. In *Digest IEEE/ACM Intl. Conf. on Computer-Aided Design*, pages 568–571, 1991.

[12] K. Karplus. Xmap: A technology mapper for table-lookup FPGAs. In *Proc. ACM/IEEE Design Automation Conf.*, pages 240–243, 1991.

[13] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[14] A. Mathur and C. L. Liu. Performance driven technology mapping for lookup-table based FPGAs using the general delay model. In *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1994.

[15] AT&T Microelectronics. *AT&T Field-Programmable Gate Arrays Data Book*. AT&T Microelectronics, 1995.

[16] R. Murgai, R.K. Brayton, and A. Sangiovanni-Vincentelli. Sequential synthesis for table look up programmable gate arrays. In *Proc. ACM/IEEE Design Automation Conf.*, pages 224–229, 1993.

[17] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli. Logic synthesis algorithms for table look up programmable gate arrays. In *Proc. ACM/IEEE Design Automation Conf.*, pages 620–625, 1990.

[18] P. Pan. Continuous retiming: algorithms and applications. In *Intl. Conf. on Computer Design (ICCD)*, pages 116–121, 1997.

[19] P. Pan and C. L. Liu. Optimal clock period FPGA technology mapping for sequential circuits. In *ACM/IEEE Design Automation Conf. (DAC)*, pages 720–725, 1996.

[20] P. Pan and C. L. Liu. Technology mapping of sequential circuits for LUT-based FPGAs for performance. In *ACM/SIGDA Intl. Symposium on Field-Programmable Gate Arrays*, pages 58–64, 1996.

[21] P. Pan and C.L. Liu. Optimal clock period FPGA technology mapping for sequential circuits with retiming. *ACM Trans. on Design Automation of Electronic Systems.* (to appear).

[22] P. Sawkar and D. Thomas. Area and delay mapping for table-look-up based field programmable gate arrays. In *Proc. ACM/IEEE Design Automation Conf.*, pages 368–373, 1992.

[23] P. Sawkar and D. Thomas. Performance directed technology mapping for look-up table based FPGAs. In *Proc. ACM/IEEE Design Automation Conf.*, pages 208–212, 1993.

[24] M. Schlag, J. Kong, and P.K. Chan. Routability-driven technology mapping for lookup table-based FPGA's. *IEEE Trans. on Computer-Aided Design*, 13:13–26, 1994.

[25] U. Weinmann and W. Rosenstiel. Technology mapping for sequential circuits based on retiming techniques. In *Proc. European Design Automation Conf.*, pages 318–323, 1993.

[26] N.-S. Woo. A heuristic method for FPGA technology mapping based on the edge visibility. In *Proc. ACM/IEEE Design Automation Conf.*, pages 248–251, 1991.

[27] Xilinx. *The Programmable Gate Arrays Data Book*. Xilinx, San Jose, CA, 1993.

[28] H. Yang and D. F. Wong. Edge-Map: Optimal performance driven technology mapping for iterative LUT based FPGA designs. In *Digest IEEE/ACM Intl. Conf. on Computer-Aided Design*, pages 150–155, 1994.