Optimal Clock Period FPGA Technology Mapping for Sequential Circuits*

Peichen Pan
Dept. of Electrical & Computer Eng.
Clarkson University
Potsdam, NY 13699

C. L. Liu
Dept. of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract – In this paper, we study the technology mapping problem for sequential circuits for LUT-based FPGAs. Existing approaches map the combinational logic between flip-flops (FFs) while assuming the positions of the FFs are fixed. We study in this paper a new approach to the problem, in which retiming is integrated into the technology mapping process. We present a polynomial time technology mapping algorithm that can produce a mapping solution with the *minimum* clock period while assuming FFs can be arbitrarily repositioned by retiming. The algorithm has been implemented. Experimental results on benchmark circuits clearly demonstrate the advantage of our approach. For many benchmark circuits, our algorithm produced mapping solutions with clock periods not attainable by a mapping algorithm based on existing approaches, even when it employs an optimal delay mapping algorithm for combinational circuits.

1 Introduction

A look-up table (LUT) based FPGA consists of an array of programmable logic blocks together with programmable interconnections [17]. The core of a programmable logic block is a k-input LUT (k-LUT) which can implement any combinational logic with up to k inputs and a single output, where k is a small positive integer. There are also several flip-flops (FFs) in each programmable logic block which can be connected to the inputs and outputs of its 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 LUTs. This problem has been studied extensively. However, almost all proposed mapping algorithms are designed for combinational circuits. Mapping algorithms for combinational circuits (will be referred to as combinational mapping algorithms from now on) have been proposed for different optimization criteria: performance [2, 6, 9, 18], area [4, 5, 7, 11, 16], routability [1, 14], and combinations of these [3, 13]. In particular, Cong and Ding [2] proposed an optimal delay combinational mapping algorithm for the unit delay model and Yang and Wong [18] proposed an optimal combinational mapping algorithm for the general delay model.

Existing approaches to technology mapping for sequential circuits use combinational mapping algorithms to map the combinational logic between FFs. These approaches have two obvious shortcomings: (i) failing to consider signal dependencies across FF boundaries, and (ii) not considering the possibility of exposing the combinational logic between FFs in different ways. Note that FFs in a sequential circuit can be repositioned by a technique called retiming [8]. Two recent sequential circuit technology mapping methods [10, 15] also assume the initial positions of the FFs are fixed, though retiming is used as a post-processing step in [15].

In this paper, we study a new approach to sequential circuit technology mapping, proposed in [12]. In this approach the FF positions are assumed to be fully dynamic in the sense that they can be arbitrarily repositioned by retiming. Our main objective is to obtain mapping solutions with minimized clock period, which is the maximum number of LUTs between any two successive FFs. We will present an efficient (polynomial time) algorithm that produces a minimum clock period mapping solution for any sequential circuit[1].

2 The new approach

To further motivate the new approach, let us examine two examples. Consider the circuit in Figure 1(a). Suppose that we want to map it to an FPGA architecture in which each LUT has at most 3 inputs. One possible mapping solution, without repositioning the FFs, is shown in Figure 1(a), where the gates enclosed by a dashed circle are mapped to one LUT. Figure 1(b) shows the mapping solution in terms of LUTs. This mapping solution uses two LUTs and has a clock period equal to two. Also note that the clock period of this mapping solution cannot be further reduced by retiming. Actually, it can be shown that any mapping solution must use at least two LUTs and have a clock period two no matter what combinational mapping algorithm is used. However, if gate b is retimed by a value one (the FF f at the output of b is moved to its inputs) as shown in Figure 1(c), all the gates can be mapped to one 3-LUT as shown in Figure 1(d). Note

*The work was partially supported by the National Science Foundation under grant MIP-9222408.

[1] The algorithm has been extended to the general delay model in which case, it produces a mapping solution with a clock period provably close to minimum.

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

that this mapping solution has a clock period of one.

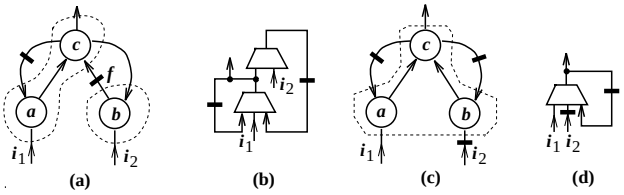

Figure 1: Advantage of retiming.

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

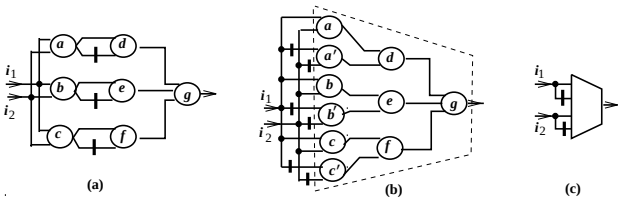

Figure 2: Advantage of logic replication.

Based on the above observations, we study the technology mapping problem in the most general setting in which the techniques of retiming and replication are exploited. Conceptually, the solution space that will be explored can be described by the diagram in Figure 3. Namely, the mapping solution space consists of all the circuits that can be obtained by retiming and replicating the given initial circuit, then mapping the combinational logic between FFs, followed by another retiming and replication step[2]. It is obvious that the solution space explored in this new approach is enormous since there are too many ways to retime and replicate a circuit.

3 Preliminaries and problem definition

A (synchronous) 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

[2]A technology mapping algorithm based on existing approaches may try to alleviate its drawbacks by carrying out these conceptual steps in sequence. However, as long as it actually employs a combinational mapping algorithm, the same drawbacks are still there.

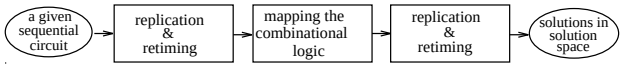

Figure 3: Solution space explored in the new approach.

combinational processing elements (PEs) in the circuit. (A PE is either a gate or a LUT depending on whether the circuit is the initial one or a mapping solution.) The edges are 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. The *clock period* of a circuit is the maximum number of PEs on the combinational paths (paths without FFs) in the circuit.

Retiming is a technique of repositioning the FFs in a circuit without changing its functionality or the structure [8]. Retiming a node by a value i means removing i FFs from each fanout edge and adding i FFs to each fanin edge of the node. Figure 4 shows the case in which $i = 1$ or -1. In general, the nodes in a circuit can be retimed collectively (referred to as retiming the circuit). It can be shown that the retimed circuit and the original one have the same functionality if no retiming is performed at the PIs and POs (i.e., the retiming values for the PIs and POs are all zero).

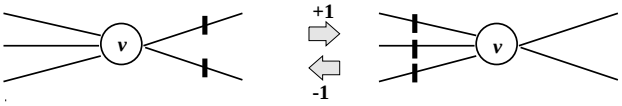

Figure 4: Retiming a node.

Refer again to Figure 3. We use N to denote the circuit to be mapped. We assume that N is *k-bounded*, namely, each node in N has at most k fanins. We will use $w(e)$ to denote the weight of an edge e in N. Let N' be a circuit obtained from N by replication and retiming and N'' be a mapping solution of the combinational logic of N'. Let S be the circuit obtained from N'' by putting the FFs back and followed by another retiming and replication. (Note that the PEs in N'' are LUTs.) S is then a *mapping solution* of N. The technology mapping problem addressed in this paper is as follows:

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

Finally, we list several graph-theoretic concepts. In a directed acyclic graph with one sink but possibly several sources, a *cut* $(X, \overline{X})$ is a partition of the nodes such that the sink is in $\overline{X}$ and all the sources are in X. The *edge-set* $E(X, \overline{X})$ of the cut is the set of edges from X to $\overline{X}$, the *node-set* $V(X, \overline{X})$ is the set of nodes in X that are connected to one or more nodes in $\overline{X}$. If $|V(X, \overline{X})| \leq k$, $(X, \overline{X})$ is called a *k-feasible* cut, or k-cut for short.

4 Formation of LUTs

In this section, we will present a method for forming LUTs for nodes in a sequential circuit.

Although the formation of LUTs is rather straightforward for combinational circuits, it is complicated for sequential circuits because a circuit may be arbitrarily retimed and replicated in the new approach. In other words, we are working with a family of circuits. To overcome this difficulty, we introduce the concept of *expanded circuits*. Our LUT formation procedure will be carried out on the expanded circuits.

An expanded circuit is constructed by replication and it has the property that all paths from any given node to the only output node have the same number of FFs.

The expanded circuits for a node v are defined recursively as follows: As the base case, the circuit with one node v^0 but no edge is an expanded circuit. Suppose $\mathcal{E}$ is an expanded circuit. Let I be the set of sources (nodes with indegree 0) in $\mathcal{E}$. We pick a node in I, say u^d. Then, for each edge $x \xrightarrow{e} u$ in N, add a node x^{d_1} where $d_1 = d + w(e)$ to $\mathcal{E}$ if it is not there, and add an edge $x^{d_1} \xrightarrow{e'} u^d$ with weight $w(e)$ to $\mathcal{E}$. The resulting circuit is also an expanded circuit.

An important class of expanded circuits consists of: $\mathcal{E}_v^i$, for $i \geq 0$. $\mathcal{E}_v^i$ denotes the expanded circuit in which the shortest distance (in terms of the number of edges) from each source that is not a replicate of a PI, to v^0 is i.

For the circuit in Figure 1(a), Figure 5 shows five expanded circuits for node c. From (a) to (e) each expanded circuit is constructed from the preceding one by expanding the shaded node. Actually, the circuit in (a) is $\mathcal{E}_v^0$, in (b) is $\mathcal{E}_v^1$, in (d) is $\mathcal{E}_v^2$, and in (e) is $\mathcal{E}_v^3$.

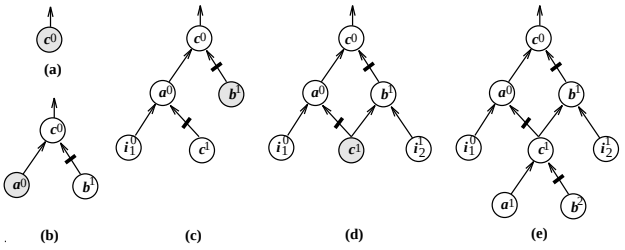

Figure 5: Expanded circuits.

We now show that a LUT for a node can be derived from a cut in the expanded circuits for the node. Let $(X, \overline{X})$ be a k-cut in an expanded circuit $\mathcal{E}$ for v. We first notice that all FFs inside $\overline{X}$ can be moved out by retiming. The retiming is: for each node u^d in $\overline{X}$, its retiming value is d; for all nodes in X, their retiming values are zero. Let $u^d \xrightarrow{e'} x$ be an edge in $E(X, \overline{X})$. (Note that u^d is a replicate of node u in N and e' is a replicate of edge e.) It can be verified that the number of FFs on e' is d after the retiming. The LUT derived from this cut is simply the subcircuit induced by the nodes in $\overline{X}$ with the FFs being removed. Let $\mathcal{L}$ denote the LUT. If u^d is in the node-set $V(X, \overline{X})$ of the cut, it means that u after passing through d FFs is an input to $\mathcal{L}$. As a result, the number of inputs to $\mathcal{L}$ is equal to the number of nodes in $V(X, \overline{X})$, which is k. Therefore, $\mathcal{L}$ is a k-LUT.

As an example, for the 3-cut indicated in the expanded circuit in Figure 5(d) as shown in Figure 6(a), Figure 6(b) shows the corresponding 3-LUT. The inputs to this 3-LUT are i_1, c passing through a FF, and i_2 passing through a FF.

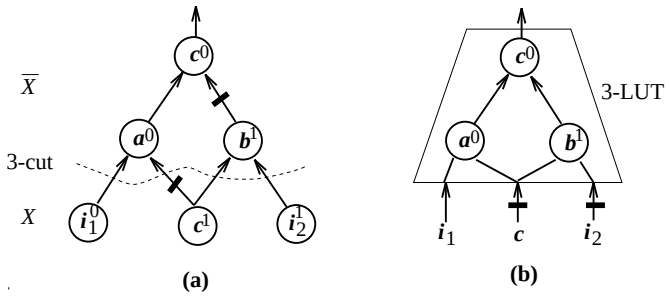

Figure 6: Derivation of a LUT from a cut.

Moreover, we can show that for any k-LUT there is a k-cut that can derive the LUT in this fashion, if the expanded circuit is $\mathcal{E}_v^{kn}$, where n is the number of nodes in N. Therefore, we have the following main result of this section:

Theorem 1 *It suffices to examine the k-LUTs for v that can be derived from the k-cuts in $\mathcal{E}_v^{kn}$.*

It can be shown the number of nodes in $\mathcal{E}_v^i$ is $O(ni)$ and the number of edges is $O(kni)$. In particular, the numbers of nodes and edges in $\mathcal{E}_v^{kn}$ are $O(kn^2)$ and $O(k^2n^2)$, respectively.

5 An algorithm for finding an optimal mapping solution

The way we solve Problem 1 is to solve its decision version as stated in the following:

Problem 2 *Given a target clock period ϕ, determine a mapping solution with a clock period of ϕ or less, whenever such a mapping solution exists.*

If we can solve Problem 2, we can do a binary search on ϕ to find a mapping solution with the minimum clock period.

We describe our algorithm for solving Problem 2 in this section. The algorithm has two phases: the labeling phase and the mapping phase. In the labeling phase, we compute a label (defined later) for each node in N. After we have computed all the labels and determined that there is a mapping solution with a clock period of ϕ or less, we then generate one such mapping solution in the mapping phase. In the next two subsections, we present the details of the two phases, separately.

5.1 The labeling phase

Let S be a mapping solution. We define a value (called l-value) for each LUT in S. To define the l-values, we use a graph whose topology is the same as that of S, and assign a weight $-\phi \cdot w_1(e) + 1$ to an edge e, where $w_1(e)$ is the number of FFs on e in S. The *l-value* of a LUT in S is the maximum weight of the paths from the PIs to the LUT according to the new edge weights.

The *label* of a node in N is the *minimum* of the l-values of the k-LUTs for the node, generated according to Theorem 1.

For a node v in N, we will use $l^{opt}(v)$ to denote its label. We determine $l^{opt}(v)$ for each node v in N in this phase of the algorithm.

Our method for computing the labels is quite similar to a longest path algorithm. The approach is to compute a lower-bound on the value of each label and to repeatedly improve (increase) the lower-bound. The lower-bounds will be equal to the actual labels when no further improvement is observed for all the lower-bounds. Initially, the lower-bound for all PIs are zero and the lower-bounds for all other nodes are $-\infty$. Figure 7 shows the overall algorithm, where $l(v)$ denotes the lower-bound on $l^{opt}(v)$. Improving the lower-bounds is carried out by Procedure IMPROVE.

```
L_FIND(N, φ)
// V denotes the set of nodes in circuit N,
// w(e) denotes the number of FFs on edge e in N
1.    for each node v in V    // initialization
2.        if v is a PI
3.            then  l(v) ← 0;
4.            else  l(v) ← −∞;
5.    updated ← FALSE;
6.    for i ← 1 to |V|    // improve at most n times
7.        for each node v in V
8.            if IMPROVE(v) = TRUE, updated ← TRUE;
9.        if updated = FALSE, return success;
10.       updated ← FALSE;
11.   return failure;

IMPROVE(v)
a.    Determine l_new;
b.    if l(v) < l_new
c.        then
d.            l(v) ← l_new ;
e.            return TRUE;    // improved
f.    return FALSE;    // not improved
```

Figure 7: Algorithm for computing the labels.

The purpose of Procedure IMPROVE is to test whether we can improve the current lower-bound on the label of v, based on the current lower-bounds on all labels, and if so, to update the current lower-bound for v. l_{new} is the new lower-bound for v computed from the current lower-bounds.

Now the remaining issue is to determine l_{new}. Based on the discussion in Section 3, we have

$$l_{new} = \min_{(X,\overline{X})} \left(\max\{l(u) - \phi \cdot d + 1 \mid u^d \text{ is in } V(X,\overline{X})\}\right),$$

where the minimum is taken over all k-cuts in $\mathcal{E}_v^{kn}$.

We will use the above formula to compute l_{new}. Our approach is to study the corresponding decision problem, namely,

Problem 3 *Check whether $l_{new} \leq L$ for a given integer L.*

We use network flow techniques to solve Problem 3. 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* to reduce the problem of finding a k-cut to that of finding a cut with an edge 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 supersource is added and connected to all the sources. The bridging edge for node u^d has capacity one if $l(u) - \phi \cdot d + 1 \leq L$. All other edges in G has infinite capacity.

As an example, suppose for the circuit in Figure 1(a), 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. In the expanded circuit for c in Figure 8(a), suppose we want to test whether $l_{new} \leq 1$. For node b^1, $l(b) - \phi \cdot 1 + 1 = 1$, so the corresponding bridging edge has capacity one. On the other hand, for node a^0, $l(a) - \phi \cdot 0 + 1 = 2$, so the corresponding bridging edge has infinite capacity. Figure 8(b) 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.

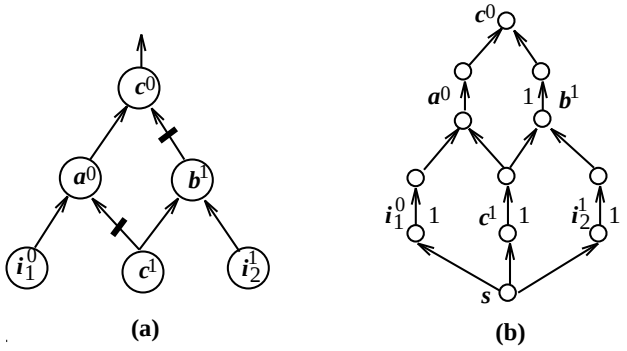

Figure 8: Construction of flow network.

The edge capacity of a cut is the sum of the capacities of the edges in the edge-set of the cut. The following result can be shown for the flow network G:

Lemma 1 *$l_{new} \leq L$ iff G has a cut with edge capacity no more than k.* □

Based on the classical Max-flow Min-cut Theorem, G has a cut with edge capacity no more than k iff the maximum flow in G is at most k. We can, therefore, use an augmenting path algorithm for solving the max-flow problem to determine whether G has a cut with edge capacity no more than k in $O(k \cdot |E(G)|) = O(k^3 n^2)$ time. Thus, we can determine whether $l_{new} \leq L$ in $O(k^3 n^2)$ time.

Obviously, l_{new} is from the following set

$$\{l(u) - \phi \cdot d + 1 \mid u^d \text{ is in } \mathcal{E}_v^{kn}\}$$

whose size is $O(kn^2)$, the number of nodes in $\mathcal{E}_v^{kn}$. We can first sort all the values in the set , and then use binary search to determine l_{new}. Overall, we have an $O(k^3 n^2 \log(kn))$-time algorithm for determining l_{new}.

L_FIND(N, ϕ) needs to call Procedure IMPROVE $O(n^2)$ times in the worst case. In summary, we have the following result:

Theorem 2 *The labels of all nodes in N can be determined in $O(k^3n^4\log(kn))$ time.* □

Remark: To guarantee that a mapping solution with the target clock period can always be found whenever there exists one, we need to use the expanded circuit $\mathcal{E}^{kn}$. This is the worst case scenario. In practice, we may use an expanded circuit $\mathcal{E}^i$ for an i considerably smaller than kn. For instance, for node c in the circuit in Figure 1(a), it can be shown that it is sufficient to use $\mathcal{E}_c^2$ (in Figure 5(d)) for examining the 3-LUTs for c. To make our algorithm flexible and to save computation time, we can use i as a control parameter so that the expanded circuit $\mathcal{E}^i$, instead of $\mathcal{E}^{kn}$ is used in Procedure IMPROVE.

5.2 The mapping phase

The purpose of this phase is to generate a mapping solution with a clock period of ϕ or less (if, of course, there is one such mapping solution).

The first step is to assemble a mapping solution from the LUTs corresponding to the cuts that realize the labels. To do so, we trace from the POs backward and to include those LUTs that are on paths from PIs to POs in the mapping solution. Specifically, we keep two lists D and U. D is the set of nodes in N whose k-LUTs have already been included in the partial mapping solution and U is the set of nodes whose k-LUTs are inputs to some k-LUTs in D and have not yet been included in the partial mapping solution. At the beginning, D consists of the PIs and U consists of the POs. At each iteration, a node v in U is removed and added to D. Let the k-LUT that realizes $l^{opt}(v)$ be $\mathcal{L}_v$ which is determined in the labeling phase. Then, if u after passing d FFs is an input to $\mathcal{L}$ we create an edge from $\mathcal{L}_u$ to $\mathcal{L}_v$ with weight d in S, and add u to U if it is not in D or U. This process stops when U becomes empty. Let S denote the resulting mapping solution. After the process is finished, D may not contain all the nodes in N. For those nodes not in D, they disappear because they are contained in some of the LUTs.

We now define a retiming r on S. For each LUT $\mathcal{L}_v$ in S, the retiming value is as follows:

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

Let S_r denote the circuit obtained from S by applying retiming r. Note that by definition S_r is also a mapping solution of N. We have the following result:

Theorem 3 *The following three statements are equivalent:*

(i) *N has a mapping solution with a clock period of ϕ or less.*

(ii) *$l^{opt}(v) \leq \phi + 1$ for each PO v in N.*

(iii) *S_r has a clock period of ϕ or less.* □

Based on Theorem 3, we can check whether there is a PO whose label is larger than $\phi + 1$ after the labeling phase. If this is the case, the algorithm will not proceed to the mapping phase because there is simply no mapping solution with the target clock period ϕ. If for each PO, its label is less than or equal to $\phi + 1$, the algorithm simply return S_r since it meets the target clock period ϕ.

6 Experimental results

Our optimal clock period mapping algorithm has been implemented in the C language (referred to as SeqMapII). Experiments were carried out on sequential benchmark circuits in the LGSynth91 suite. In this section, we describe our experiments and summarize the results.

For comparison, we also implemented a technology mapping algorithm based on existing approaches which will be referred 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 [2]. ComMap also uses retiming as a pre-processing step as well as a post-processing step. Specifically, it retimes the initial circuit to the minimum clock period before applying FlowMap. It also retimes the mapping solution to the minimum clock period after FlowMap. The resulting circuit is then the output of ComMap.

We tested both ComMap and SeqMapII on a set of benchmark circuits using 5-LUTs. The results are summarized in Table 1. In Table 1, under column **initial** we list the number of gates and the number of FFs of each benchmark circuit (decomposed using *tech_decomp -a 2 -o 2* in **SIS**). 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 SeqMapII. For SeqMapII, we set the control parameter i, the depth of the expanded circuits used to form LUTs, to be 6 in the experiments. (Therefore, the clock periods of the mapping solutions produced by SeqMapII might not be minimum.) Even with such a small depth, SeqMapII consistently produced mapping solutions with smaller clock periods than that produced by ComMap as can be observed from the table. This clearly shows the advantage of the new approach. It can also be seen that the mapping solutions produced by SeqMapII usually have fewer LUTs than that produced by ComMap. This was also expected since SeqMapII can form LUTs by extending across FF boundaries. Overall, the mapping solutions produced by ComMap use 10% more LUTs, 33% larger clock periods, and 2% less FFs. For all the test circuits except s38417, the CPU times of our current implementation of SeqMapII were less than two minutes and in most cases only a few seconds on a SPARC 5 workstation with 32Mb memory. However, for s38417 it took SeqMapII close to 30 minutes due to the large size of the circuit and a larger reduction in the clock period. Overall, the CPU times of SeqMapII are about 10 times that of ComMap for the test circuits.

7 Conclusions

In this paper, we studied the FPGA technology mapping problem for sequential circuits in the most general setting. In our approach, retiming is fully integrated into the mapping process. As a result, the mapping solution space explored by our approach is much larger than what existing approaches are able to explore. Another way to understand our approach is that for our approach, there is no FF boundary at all, in the sense that if one circuit is obtained from another one by retiming, our algorithm will produce the same mapping solution for both circuits. In other words, where to place the

test circuit	Initial		ComMap			SeqMapII		
	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

Table 1: Experimental results.

FFs in a circuit has no effect on our algorithm. On the other hand, for a mapping algorithm based on existing approaches, there always exist FF boundaries and signal dependencies across FF boundaries are severed. We further presented a polynomial mapping algorithm which can produce mapping solutions with the minimum clock periods.

References

[1] N. Bhat and D. Hill. Routable technology mapping for FPGAs. In *ACM/SIGDA Workshop on FPGAs*, pages 143–148, 1992.

[2] 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.

[3] 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.

[4] 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.

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

[6] R. J. Francis, J. Rose, and Z. Vranesic. Technology mapping for lookup table-based FPGAs for performance. In *Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 568–571, 1991.

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

[8] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. In *Proc. 3rd Caltech Conf. on VLSI*, pages 87–116, 1983.

[9] 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.

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

[11] R. Murgai, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli. Improved logic synthesis algorithms for table look up architectures. In *Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 564–567, 1991.

[12] 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.

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

[14] 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.

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

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

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

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