DELAY-OPTIMAL TECHNOLOGY MAPPING BY DAG COVERING

by

Yuji Kukimoto, Robert K. Brayton, and Prashant Sawkar

Memorandum No. UCB/ERL M97/75

10 October 1997

DELAY-OPTIMAL TECHNOLOGY MAPPING BY DAG COVERING

by

Yuji Kukimoto, Robert K. Brayton, and Prashant Sawkar

Memorandum No. UCB/ERL M97/75

10 October 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Delay-Optimal Technology Mapping by DAG Covering

Yuji Kukimoto Robert K. Brayton Prashant Sawkar†
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720
Strategic CAD Laboratories, Intel Corporation
Hillsboro, OR 97124†
{kukimoto,brayton}@eecs.berkeley.edu
psawkar@ichips.intel.com

October 10, 1997

Abstract

We propose an optimal algorithm for delay minimal technology mapping for library-based designs. We show that subject graphs need not be decomposed into trees for delay minimization; they can be mapped directly as DAGs. Experimental results demonstrate that significant delay improvement is possible by this new approach.

1 Introduction

Logic synthesis typically consists of two phases. The first step, called *technology-independent optimization*, is the step in which a given circuit is restructured without knowing an actual gate library or technology to be used. Generic optimization such as factoring, resubstitution and don't care minimization is performed to seek a good multi-level structure. This step is followed by *technology mapping*, in which the optimized circuit in the previous step is implemented by only using gates in a given library. The importance of technology mapping is increasing significantly since it is very difficult in deep sub-micron designs to estimate the effect of a generic optimization without knowing an actual technology to be used.

Technology mapping was initially done by rule-based transformations in early 80's. The approach is ad-hoc and has no guarantee about mapping quality. Furthermore different sets of transformation rules need to be maintained for different libraries. In 1987 Keutzer [8] proposed an algorithmic approach to the technology mapping problem, in which he observed similarity between this problem and the code optimization problem

for programming languages and adapted an existing tree-covering technique for the latter problem to technology mapping. This approach soon dominated the rule-based approach and became the de facto standard.

In Keutzer's formulation a technology-independent circuit and each gate in a given library are decomposed into NAND2-INV circuits. The decomposed circuit is called a *subject graph* while each decomposed gate is called a *pattern graph*. The technology mapping problem can then be formulated as covering the subject graph by using pattern graphs. A subject graph is a directed acyclic graph (DAG) in general since it is derived from a given network. Keutzer showed that if a subject graph is a DAG, graph covering for *minimum area mapping* is NP-hard [9]. He further proposed a linear-time dynamic programming algorithm which guarantees optimal results for the case where a subject graph and pattern graphs are trees. These observations led naturally to the following three-step approach.

1. Decompose a subject DAG into a disjoint set of trees
2. Solve the technology mapping problem optimally for each tree
3. Glue the results together.

This separation of the problem again has become a standard approach due to the theoretical justification about the complexity of DAG covering.

Inspired by Keutzer's result technology mapping has been studied extensively to optimize different criteria. Rudell [13] worked on *minimum-delay* technology mapping and showed that if loading effects are completely ignored, the minimum-delay mapping problem for subject *trees* can be solved optimally by dynamic programming in linear time. He also considered the minimum-delay mapping problem for trees under loading effects and showed that by maintaining the best mapping for each possible load at each node the same dynamic programming approach can guarantee optimal results. Touati [14] further refined this idea later by combining the optimal tree mapping with sophisticated buffer tree construction. An interesting fact is that they directly started looking at *tree covering* without studying the complexity of *DAG covering* for minimum delay. This was even true for a more recent work by Lehman and Watanabe [10].

In parallel to these works on library-based technology mapping the emergence of FPGAs posed a new technology mapping problem in early 90's. LUT-based FPGAs can implement any function of k inputs by a single LUT, where k is a fixed constant depending on a given technology. It is not practical to follow the same approach as library-based technology mapping since one needs to generate pattern graphs for all 2^{2^k} k-input functions. Based on this observation many ideas have been proposed for the FPGA mapping problem again under different cost criteria [4]. As for minimum area mapping Levin *et al.* [11] and Farrahi *et al.* [7] proved that the problem is NP-hard for $k = 4$ and $k \geq 5$ respectively. Minimum-delay mapping, on the other hand, was shown for LUT-based FPGAs to be solvable in polynomial time by Cong *et al.* in [1, 2]. Here the given circuit is directly mapped without decomposing its DAG structure to trees in this algorithm unlike conventional library-based mapping.

In this paper we consider the minimum-delay technology mapping problem for library-based designs where a subject graph is a DAG. Careful analysis of [2] shows that the basic dynamic programming approach in [2] is not specific to FPGA mapping and can be easily adapted to library-based mapping. This leads to a linear time algorithm for minimum-delay DAG covering under load-independent delay models. As far as we know, this is the first result that shows that the minimum-delay technology mapping problem for DAGs can be solved optimally in polynomial time. This implies that tree decomposition in performance-oriented mapping is not necessary; a given subject DAG can be directly mapped optimally. We experimentally confirmed that the additional solution space explored by this direct approach finds significantly faster mappings especially under a rich library.

This paper is organized as follows. Section 2 reviews library-based technology mapping and recent work on delay-optimal FPGA mapping. The optimal delay mapping algorithm for FPGAs presented in [2] is overviewed. Based on this algorithm Section 3 shows that the basic algorithmic principle of [2] is not limited to FPGAs and discusses how the algorithm can be adapted to library-based technology mapping. Section 4 discusses an extension of this idea to sequential circuits under consideration of retiming. A simple modification to an existing FPGA technology mapping algorithm for minimum-cycle time [12] leads to an algorithm for optimal cycle-time library-based technology mapping. An idea of combining this work with Lehman *et al.*'s work [10] is also discussed. Experimental results are shown in Section 5 to demonstrate the effectiveness of this DAG covering approach compared with the traditional tree covering. Section 6 concludes the paper with future directions of this work.

2 Preliminaries

2.1 Library-Based Technology Mapping

We will briefly overview basic concepts of library-based technology mapping. We will closely follow a strategy proposed by Keutzer [8].

Given a combinational Boolean network, it is first decomposed into a NAND2/INV network by decomposing each node in the network into a NAND2/INV structure. The resulting network, called a *subject graph*, is a Boolean network in which every node is either a 2-input NAND or an inverter. Each gate in a given library is similarly decomposed into a NAND2/INV network, called a *pattern graph*. Typically all possible NAND2/INV decompositions are generated for each gate modulo isomorphism so that the gate is utilized maximally in a final implementation. Each pattern graph is associated with the area, delay and other characteristics of the corresponding gate. Once this decomposition step is done, the subject graph is covered by using pattern graphs to optimize a given cost criterion.

Since a subject graph is constructed from a given Boolean network, it is usually a directed acyclic graph (DAG). A pattern graph is a tree for most gates in a typical library while it can be a DAG for some gates, e.g. an XOR gate and a multiplexor gate.

For the sake of simplicity assume that all the pattern graphs are trees for now.

Keutzer investigated the computational complexity of the minimum-area technology mapping problem in [9] and proved that if a subject graph is a DAG, the problem is NP-hard. Having demonstrated the inherent complexity of the original problem, he considered the case where a subject graph and pattern graphs are trees. It turned out that this special case can be solved optimally in linear time using dynamic programming. Based on these results he proposed the following three-step approach as an approximation to the DAG covering problem.

1. Decompose a subject DAG into a disjoint set of trees
2. Solve the minimum-area mapping problem optimally for each tree
3. Glue the results together.

Since this strategy using tree covering became so common, the same approach has been also employed in optimizing delay, but interestingly enough to the best of our knowledge no one investigated the exact complexity of the minimum-delay technology mapping problem where a subject graph is a DAG. Probably it was simply assumed that the problem is NP-hard without giving much thought.

Now consider the case where some pattern graphs are DAGs. Rudell showed that as long as those are leaf DAGs, the tree covering approach can be used without any modification [13]. A leaf DAG is a DAG in which the only nodes with multiple fanouts are primary inputs. An XOR gate and a multiplexor gate have leaf DAG pattern graphs and thus can be handled without any problem.

So far we have focused on the case where a subject graph is a tree. To conclude this subsection we review previous work on DAG covering without tree decomposition. Detjens *et al.* studied this problem for area minimization in [6], but since a heuristic approach was taken for covering, the results were not impressive. In fact the DAG covering approach gave results of lesser quality than the tree-based approach. Although they also described an idea on node duplication similar to [2] to be detailed later, it was apparently only tried for area optimization and no results are reported on this approach in [6].

2.2 Technology Mapping for LUT-based FPGAs

Due to their unique architecture the technology mapping problem for FPGAs has been tackled in completely different ways from library-based technology mapping.

In LUT-based FPGAs each LUT can implement any function of k-inputs in the same delay, where k is a constant specific to a given FPGA family. By assuming the existence of a library containing all k-input functions, one can solve the technology mapping problem for FPGAs as an instance of the library-based mapping problem. This approach, however, has a serious drawback since the library constructed this way

contains 2^{2^k} gates[1]. Even if the minimum-area tree covering for library-based designs can be solved in time *linear* in the size of pattern graphs, the number of gates makes the algorithm highly inefficient.

The minimum-area technology mapping problem for LUT-based FPGAs was shown to be NP-hard for $k = 4$ in [11] and for $k \geq 5$ in [7]. As in the library-based mapping, once a network is restricted either to a tree, the problem can be solved optimally in polynomial time [7].

The minimum-delay technology mapping problem, however, has a different story. Cong *et al.* showed in [1, 2] that the problem is solvable in polynomial time even for DAG networks.

We will have a close look at the FlowMap algorithm presented in [2] next since this gives the basis of our proposed algorithm.

2.3 Delay-Optimal Technology Mapping for FPGAs

Assume that a network is decomposed into a k-bounded network [2], which is a Boolean network where the number of fanins of each node is less than or equal to k. If a given network is not k-bounded, simple decomposition can yield an equivalent k-bounded network. In the following we assume that an LUT has a unit delay and that wiring delay is negligible.

The key idea of the FlowMap algorithm is in the labeling procedure that labels each node of the DAG its optimal depth achievable. The algorithm visits each node in the network in a topological order. All primary inputs are labeled 0 assuming that they are available at $t = 0$. At each intermediate node the goal is to investigate all cuts of size less than or equal to k in the fanin cone of the node and to find the best delay realizable at the node. Each such cut represents a mapping of the node. More specifically the node can be implemented by a single LUT whose inputs are the nodes forming the cut. The constraint on the size of cuts comes from the fact that an LUT can implement any function of up to k-inputs. Since nodes are visited in a topological order, by the time the current node is examined the optimal depths of all the nodes in its transitive fanin are available. Therefore the optimal depth of the current node x can be computed as follows by dynamic programming.

$$optimal\_depth(x) = \min_{\text{cut } X:|X|\leq k} \max_{x_i \in X} (optimal\_depth(x_i) + 1)$$

Notice that this cost criterion meets *the principle of optimality* of dynamic programming. The cut X that realizes the optimal depth is stored at the node along with the depth. Although explicit enumeration of all valid cuts is possible by a brute-force approach, the complexity is pseudo polynomial $O(n^k)$, where n is the number of nodes in a given network. [2] showed that this optimal depth computation at each node can be

[1] Strictly speaking the library need not contain all the 2^{2^k} functions since some are equivalent to each other under input permutation and having one representative is good enough. However even after this simplification the library is still huge.

formulated as network flow computation whose runtime is strongly polynomial with respect to k.

Once all the nodes have been labeled by their optimal depths, the network is traversed backward from primary outputs to primary inputs. At each primary output an LUT is created whose fanins are the same as the best cut stored at the node. The LUT creation is repeated for each of those fanins. This process is continued until either a primary input or a node whose output is already available in the mapping is reached. An important fact is that intermediate nodes are automatically duplicated in an optimal way to guarantee optimal depths while in tree mapping no duplication is allowed.

The complexity of the entire algorithm is $O(kmn)$, where m is the number of edges in the network.

3 Delay-Optimal Technology Mapping for Library-Based Designs

Although the FlowMap algorithm was originally developed for FPGAs, the basic principle of the labeling procedure is not necessarily specific to those architectures[2]. In this section we will show how the FlowMap algorithm can be easily adapted to the standard library-based technology mapping under a load-independent delay model, where each gate has an intrinsic delay and loading has no effect in delays. This extension leads to a linear time algorithm for delay-optimal technology mapping of DAG networks. We assume that a given network is decomposed into a subject DAG as usual. Therefore, the optimality of delay is claimed with respect to this subject DAG.

3.1 Computation of Optimal Delay at Internal Nodes

The only difference between FPGAs and library-based designs is how an internal node is mapped. In FPGAs all the local mappings that cover an internal node and part of its fanin cone are examined by enumerating all k-cuts of the fanin cone, which gives the best possible delay realized at the node. This step needs to be modified for library-based designs so that all successful matches for a given set of pattern graphs are systematically examined. However this can be easily done by mimicking the standard pattern matching step used in conventional technology mapping. More precisely once the fanin cone of the node is extracted, the standard matching procedure against pattern graphs can be applied to exhaustively check all the successful matches. This way the best delay achievable at each intermediate node can be computed in a similar way to [2]. The only difference is that actual pin-to-pin delays of gates specified in a given

[2]It is interesting to note that [2] has a comment as follows.

> "Our result makes a sharp contrast with the fact that the conventional technology mapping problem in library-based designs is NP-hard for general Boolean networks."(page 2 [2])

library need to be used in our case instead of unit delay in [2]. As with FPGA mapping, the principle of optimality is still valid here.

Notice that as long as delay is optimized, any DAG pattern graph can be used directly without losing the optimality, i.e. it is not necessary to restrict the library to pattern graphs of trees and leaf DAGs. General DAG patterns are problematic only in the context of area optimization.

3.2 Pattern Matching

We now examine how pattern matching is done between a subject graph and a pattern graph.

Pattern matching between a subject graph and a pattern graph in the context of technology mapping was studied extensively by Keutzer [8] and Rudell [13]. A match between a subject graph $G_s = (V_s, E_s)$ and a pattern graph $G_p = (V_p, E_p)$ is defined as follows [13].

Definition 1 *A (standard) match of a pattern graph $G_p = (V_p, E_p)$ on a subject graph $G_s = (V_s, E_s)$ is a* one-to-one *mapping of the pattern graph nodes into the subject graph nodes $I : V_p \mapsto V_s$ such that:*

1. $\forall e = (e_1, e_2) \in E_p, (I(e_1), I(e_2)) \in E_s$.
2. $\forall v \in V_p, |i(v)| \neq 0 \Rightarrow |i(v)| = |i(I(v))|$, *where* $i(v) = \{w \mid (w, v) \in E\}$ *for* $G = (V, E)$.

The first condition requires that the edge relationship in the pattern graph is completely preserved in the subject graph. The second condition constrains the in-degree of a non-primary-input node in the pattern graph to be the same as that of the matching node in the subject graph. Notice that it is allowed for a subject graph node covered by an intermediate pattern graph node to have fanout to nodes which are not covered by the pattern graph. However, in the conventional tree-covering based approach such a match is invalid, i.e. all fanouts of a subject graph node matched with an intermediate pattern graph node need to be covered by the same pattern. A match satisfying this additional constraint is called an *exact* match [13] and defined as follows.

Definition 2 *An exact match of a pattern graph $G_p = (V_p, E_p)$ on a subject graph $G_s = (V_s, E_s)$ is a one-to-one mapping of the pattern graph nodes into the subject graph nodes $I : V_p \mapsto V_s$ such that:*

1. $\forall e = (e_1, e_2) \in E_p, (I(e_1), I(e_2)) \in E_s$.
2. $\forall v \in V_p, |i(v)| \neq 0 \Rightarrow |i(v)| = |i(I(v))|$
3. $\forall v \in V_p, |i(v)| \neq 0$ *and* $|o(v)| \neq 0 \Rightarrow |o(v)| = |o(I(v))|$, *where* $o(v) = \{w \mid (v, w) \in E\}$ *for* $G = (V, E)$.

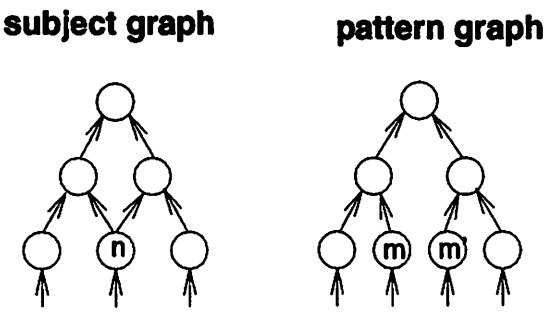

Figure 1: Standard Match vs. Extended Match

Rudell proposed an algorithm called `graph_match` [13] for the general case where both a subject graph and a pattern graph are DAGs. We can simply use this matching algorithm to enumerate all successful standard matches instead of exact matches.

Although a constraint that a mapping is *one-to-one* is posed in the above two definitions by Rudell, this is safely dropped as follows, which leads to the definition of a larger class of matches.

Definition 3 *An* extended *match of a pattern graph $G_p = (V_p, E_p)$ on a subject graph $G_s = (V_s, E_s)$ is a mapping of the pattern graph nodes into the subject graph nodes $I : V_p \mapsto V_s$ such that:*

1. $\forall e = (e_1, e_2) \in E_p, (I(e_1), I(e_2)) \in E_s$.
2. $\forall v \in V_p, |i(v)| \neq 0 \Rightarrow |i(v)| = |i(I(v))|$

The only difference between extended matches and standard matches is that in extended matches the requirement of a mapping from the pattern graph nodes into the subject graph nodes being one-to-one is dropped. Therefore extended matches subsume standard matches. This relaxation of the requirement allows duplication of subject graph nodes while searching for a match by unfolding a DAG structure. Figure 1 shows an example where a pattern graph is matched successfully as an extended match but not as a standard match. Assume that a two-input node is an NAND2 gate and a single-input node is an inverter. Consider pattern matching at the top node of the subject graph shown on the left against the pattern graph on the right. An extended match exists by mapping both m and m' to n while a standard match does not since such a mapping violates the one-to-one mapping property. A simple modification to the `graph_match` algorithm makes the algorithm search all extended matches instead of all standard matches without changing its asymptotic complexity.

3.3 Constructing an Optimum Mapping

Once a (best delay, best gate)-pair is computed at each node, a delay-optimal network can be constructed in exactly the same way as in [2]. We maintain a queue which

contains nodes to be created in the final mapping. This queue is initialized to the set of all primary outputs. A node is picked up from the queue and the best gate at the node is created in the mapping. Each fanin node of the gate is then inserted to the queue if the fanin is not a primary input and does not yet have a corresponding gate in the mapping. Once the queue becomes empty, the mapping is complete.

3.4 Complexity of DAG Mapping for Delay Minimization

An application of `graph_match` to enumerate all successful matches at a single node is $O(p)$ [13], where p is the number of nodes in the entire *unique* pattern graphs [3]. Since this procedure is called once at each node in a subject graph, the complexity of the labeling step is $O(sp)$, where s is the number of nodes in the subject graph. The final step of constructing a delay-optimal mapping only costs $O(s)$. Therefore the complexity of DAG mapping is $O(sp)$. Since p is a constant defined by a given technology, the procedure is linear in the size of a subject graph.

3.5 Comparison between DAG Mapping and Tree Mapping

In the past, performance-oriented technology mapping has been done by a combination of tree covering and buffer tree construction [14]. The fundamental limitation of this conventional tree-covering approach is that the search space is highly limited by the structure of a given subject graph since multiple-fanout points in the subject graph are completely preserved in the final results. On the other hand, since DAG mapping does not respect initial multiple-fanout points at all, it can explore a strictly larger search space. In other words multiple-fanout points are created as the result of delay optimization as we see later in Figures 2 and 3. Buffering techniques proposed in the literature can be directly used in conjunction with DAG covering to speed up such multiple-fanout points.

Another major difference is how subject graph nodes are duplicated during technology mapping. DAG mapping can duplicate subject graph nodes while creating final mappings whereas in tree mapping no duplication is allowed since each subject graph node is covered only once by a single pattern. In some sense, subject graph node duplication is limited to the buffer tree construction phase in the tree-mapping-based approach.

Figure 2 illustrates how duplication of subject graph nodes helps reduce the delay of a mapping. Consider a subject graph shown on the left. Suppose that a pattern graph on the right is available in a given library. If tree mapping is invoked on this subject graph, the pattern graph is of no use since there is no exact match between the subject graph and the pattern graph. If, on the other hand, DAG mapping is employed, the two output nodes in the subject graph are implemented as in Figure 3. The cone rooted

[3]Note that p is not equal to the number of nodes in the entire pattern graphs since during matching a single pattern graph is tried for all possible permutations of its inputs. p is thus the number of nodes in the expanded pattern graphs. See [13] for details.

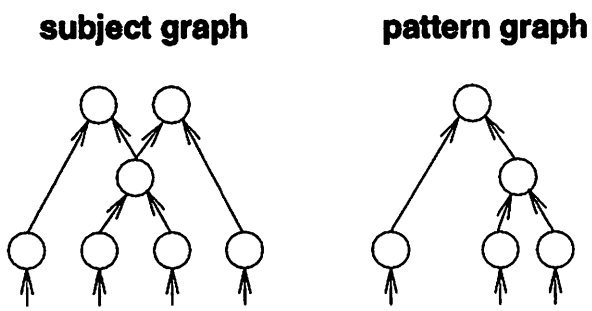

Figure 2: Example

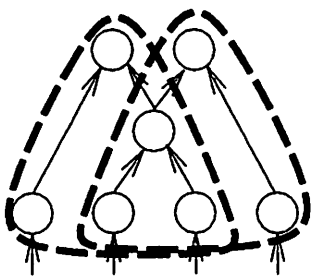

Figure 3: Duplication of Subject Graph Nodes in DAG Mapping

at the middle node in the subject graph is duplicated in this mapping, which makes effective use of the pattern graph possible.

This example also illustrates how multiple-fanout points are created in DAG mapping. Since the middle node of the subject graph with multiple fanouts is an internal node of each of the matchings in Figure 3, the mapped circuit does not inherit the multiple fanout point. On the other hand, the two primary inputs of the subject graph in the middle have multiple fanouts in the mapped circuit while each of the inputs has a single fanout in the subject graph.

4 Extensions

We will examine how the approach of Section 3 can be generalized to sequential circuits so that optimal cycle time is guaranteed in conjunction with retiming. We only consider sequential circuits with edge-triggered latches all of which are controlled by a single clock.

This problem was studied for LUT-based FPGAs by Pan and Liu [12]. Given a k-bounded network consider the following three-step transformation.

circuit	Delay		Area		CPU time	
	tree-mapping	DAG-mapping	tree-mapping	DAG-mapping	tree-mapping	DAG-mapping
C432	12.13	10.29	442	484	0.5	0.5
C499	10.16	8.03	904	960	0.9	1.1
C880	9.43	7.87	710	755	0.8	0.9
C1355	13.06	9.66	1146	1488	1.1	1.2
C1908	13.87	10.71	1223	1572	1.5	1.7
C2670	11.54	9.43	1552	2008	2.3	2.6
C3540	17.20	14.00	2075	2926	3.1	3.7
C5315	16.55	13.04	3687	4275	5.4	6.0
C6288	56.99	41.95	4107	9291	4.9	5.9
C7552	14.23	11.06	4983	6452	6.8	8.4

Table 1: Tree mapping vs. DAG mapping for `lib2.genlib`

1. Retime an initial circuit
2. Perform technology mapping of the combinational portion of the circuit
3. Retime the resulting mapped circuit.

[12] proposed a polynomial-time algorithm for computing the minimum cycle-time mapping among all the mapped circuits obtained by the above transformation, which was later improved in [5]. The key ingredient is a polynomial-time decision procedure which determines whether there exists a mapping whose cycle time is less than or equal to a given value. This procedure is used repeatedly to guide a binary search to determine the minimum cycle time achievable by retiming and optimal technology mapping. The core of this decision procedure is again a labeling scheme quite similar to the one used in FlowMap [2]. All k-cuts at each intermediate node are explored by considering retiming possibility. This is again done implicitly by converting the original problem to a flow network problem. This step of examining all k cuts can be replaced by pattern matching as was done for combinational mapping. All the other theories hold without any modification. Details are omitted here due to page limitation.

So far optimality is guaranteed in terms of a subject graph constructed arbitrarily from a given circuit by decomposition. Since a single subject graph is chosen among a huge number of different decompositions without knowing an actual library to be used, it is likely that many potentially good mappings are simply not explored due to this initial choice. Lehman *et al.* [10] have recently resolved this issue by encoding various decompositions into a single extended subject graph called *mapping graph* and performing technology mapping on it. Since this technique is orthogonal to our technique, the two can be combined to produce even better results.

5 Experimental Results

circuit	Delay		Area		CPU time	
	tree-mapping	DAG-mapping	tree-mapping	DAG-mapping	tree-mapping	DAG-mapping
C432	24	19	784	1006	0.4	0.4
C499	25	16	1772	2220	0.8	0.8
C880	20	15	1250	1337	0.7	0.7
C1355	27	22	2100	1546	1.0	1.0
C1908	37	24	2251	3058	1.3	1.3
C2670	27	18	2998	4568	2.0	2.0
C3540	42	30	4007	6640	2.7	2.8
C5315	46	33	6817	8352	4.6	4.8
C6288	125	120	7782	7121	4.3	4.4
C7552	39	28	9552	11149	6.0	6.3

Table 2: Tree mapping vs. DAG mapping for `44-1.genlib`

circuit	Delay		Area		CPU time	
	tree-mapping	DAG-mapping	tree-mapping	DAG-mapping	tree-mapping	DAG-mapping
C432	21	11	624	1094	21.5	38.5
C499	18	9	1324	1910	35.3	68.9
C880	18	8	946	1466	35.2	55.9
C1355	26	10	1796	2440	41.5	69.3
C1908	28	11	1755	2587	57.2	123.5
C2670	22	10	2314	3943	92.2	159.7
C3540	28	13	2983	6148	128.2	255.6
C5315	31	15	5115	6685	220.4	341.5
C6288	125	42	7694	14775	155.1	229.5
C7552	27	13	7062	13267	248.7	491.0

Table 3: Tree mapping vs. DAG mapping for `44-3.genlib`

To show the effectiveness of this approach the technology mapper of SIS was extended so that delay-optimal mapping is obtained for combinational circuits by DAG covering [4]. As discussed in the previous sections the delay model used in this experiment is the intrinsic delay model where a fixed, load-independent delay is given between each input and the output of a gate. This is in fact the delay model used in [10]. Although loading effects are certainly an important factor in delays, there are several justifications. In design scenarios where continuous resizing of any gate is permissible one way to capture this flexibility in technology mapping is to approximate this flexibility by having many discretely resized gates. Unfortunately this approach is known to be very expensive. The approach taken in [10] is to pick a single delay for each gate and perform technology mapping by ignoring loads. Each gate in the final mapping is then continuously resized by considering actual loads so that the delay matches the one associated with the gate. Even without the capability of continuous resizing, the buffer tree construction methods of [14] can be used later at multiple fanout points to reduce load dependency of delays. Therefore the use of this delay model is at least justified as an approximation to the minimum-delay mapping problem under realistic delay models.

Table 1 shows the comparison of the quality of final circuits between the DAG mapping approach proposed in this paper and the standard tree mapping approach. In this experiment each benchmark circuit was first decomposed into a subject graph. We then applied the DAG mapping algorithm and the tree mapping algorithm on this same subject graph using MCNC gate library `lib2.genlib`[5]. No technology independent optimization was applied to benchmark circuits before technology mapping. No fanout optimization was used. The effectiveness of the DAG mapping algorithm is clear. We could obtain significantly faster circuits. CPU time was obtained on DEC AlphaServer 8400 5/300 and is reported in seconds. The increase of CPU time from tree mapping to DAG mapping is reasonable.

The same experiment was repeated using different libraries to see how the DAG mapping algorithm performs on rich libraries. MCNC libraries `44-1.genlib` and `44-3.genlib` were used in this comparison. The former library only contains 7 gates while the latter library has 625 gates, many of which are complex gates with many inputs [6]. `44-3.genlib` is a strict superset of `44-1.genlib`. Table 2 and Table 3 summarize the results of `44-1` and `44-3` respectively. We can see that the difference in mapping quality between the DAG and tree mapping approaches is further pronounced with the use of richer libraries.

It is interesting to observe that DAG mapping can generate faster and smaller results in some cases, for examples in C1355 and C6288 in Figure 2. The reason is that more complex gates are used in DAG mapping, which leads to area effective covering in

[4] In this experiment we used `graph_match` for only finding standard matches instead of extended matches. Therefore the optimality of the results is claimed with respect to standard matches. So far we have not been able to see any major difference in mapping quality between the use of standard matches and extended matches. We expect to see a meaningful difference once a sequential mapper based on Pan's work is implemented.

[5] Each gate has a non-zero load-dependent delay specified in `lib2.genlib`. In the experiment this was simply assumed to be zero.

[6] The largest gate has 16 inputs.

some cases in spite of potential node duplication.

6 Conclusions

We have shown that the delay-optimal technology mapping problem for combinational circuits under load-independent delay models can be solved optimally in polynomial time without decomposing a subject DAG into trees. We experimentally showed that the proposed approach gives significant improvement in mapping quality compared against conventional tree mapping. Extensions of this technique to sequential circuits have also been discussed. We are currently implementing the minimum cycle-time mapping algorithm for sequential circuits. Experimental results for sequential technology mapping will be included in the final version of this paper.

In this paper we focus on delay minimization without any area consideration. Therefore at each intermediate node the fastest mapping is simply created no matter how critical the node is. By constructing slower but smaller mapping for non-critical subnetworks we can have a better control over area increase. Cong *et al.* [3] already have results on area-delay tradeoff for FPGA mapping based on the FlowMap algorithm. An extension of this idea to library-based mapping is currently being investigated.

Acknowledgments

This work is partially supported by SRC-97-DC-324. Part of the implementation was done while the first author was with Intel during the summer of 1997.

References

[1] J. Cong and Y. Ding. An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 48–53, November 1992.

[2] J. Cong and Y. Ding. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on Computer-Aided Design*, 13(1):1–12, January 1994.

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

[4] J. Cong and Y. Ding. Combinational logic synthesis for lut based field programmable gate arrays. *ACM Transanctions on Design Automation of Electronic Systems*, 1(2):145–204, April 1996.

[5] J. Cong and C. Wu. An improved algorithm for performance optimal technology mapping with retiming in LUT-based FPGA design. In *Proceedings of IEEE International Conference on Computer Design*, pages 572–578, October 1996.

[6] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Technology mapping in MIS. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 116–119, November 1987.

[7] A. Farrahi and M. Sarrafzadeh. Complexity of the lookup-table minimization problem for FPGA technology mapping. *IEEE Transactions on Computer-Aided Design*, 13(11):1319–1332, November 1994.

[8] K. Keutzer. DAGON: Technology binding and local optimization by DAG matching. In *Proceedings of 24th ACM/IEEE Design Automation Conference*, pages 617–623, June 1987.

[9] K. Keutzer and D. Richards. Computational complexity of logic synthesis and optimization. In *Proceedings of International Workshop on Logic Synthesis*, May 1989.

[10] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic decomposition during technology mapping. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 264–271, November 1995.

[11] I. Levin and R. Y. Pinter. Realizing expression graphs using table-lookup FPGAs. In *Proceedings of the European Design Automation Conference*, pages 306–311, September 1993.

[12] P. Pan and C. L. Liu. Optimal clock period FPGA technology mapping for sequential circuits. In *Proceedings of 33rd ACM/IEEE Design Automation Conference*, pages 720–725, June 1996.

[13] R. Rudell. Logic synthesis for VLSI design. Technical Report UCB/ERL M89/49, University of California, Berkeley, April 1989.

[14] H. J. Touati. Performance-oriented technology mapping. Technical Report UCB/ERL M90/109, University of California, Berkeley, November 1990.