Delay Estimation and Optimization of Logic Circuits: A Survey

Masahiro Fujita Rajeev Murgai
Advanced CAD Department
Fujitsu Laboratories of America, Inc.
Santa Clara, CA 95054, USA
e-mail: {fujita, murgai}@fla.fujitsu.com

Abstract

Logic synthesis has two stages of optimization: technology-independent and technology-dependent. This paper surveys state-of-the-art methods for estimation and optimization of delays of logic circuits at technology-independent stage. Although at this stage we cannot completely predict final delays after technology mapping, there exist reasonably accurate estimation techniques. Final delays can be reduced with optimization techniques that use such estimation.

1 Introduction

For many years, area optimization was the primary goal of logic design and synthesis. After scores of significant advances, today it seems that the field of area optimization has matured. Instead, improving the circuit timing (or performance) has now emerged as an important optimization goal in logic synthesis.

Before any attempt is made to improve the delay characteristics of a circuit implementation, it is imperative that one measures its delay first. The true delay, however, can be measured only after the circuit has been fabricated. Therefore, when the circuit is being designed, one has to be content with using **models of circuit delay**, which predict the delay of a given circuit representation. The delay model employed depends directly on the stage at which the design is being done. For instance, at the logic design stage, one popular model is the **unit delay model**, which approximates delay by the number of logic levels in the circuit. During physical design, more accurate and sophisticated models, which take into account the wiring delays, are used. Two such models are Elmore [8] and Rubinstein-Penfield-Horowitz [14]. This step of using delay models to predict the circuit delay is known as **delay estimation**.

Once the circuit delay has been estimated, circuit transformations can be applied that would reduce the delay. Typically these transformations modify the circuit structure. Delay of the modified structure is computed; if it is less than before, the modified structure can be accepted as the new implementation. This step is known as **timing optimization** (**delay optimization, performance optimization**).

This paper presents a survey of techniques for delay estimation and optimization of combinational logic circuits. Basic definitions and concepts required for the rest of the paper are presented in Section 2. Section 3 gives an overview of various delay models used at technology-independent, logic level. The state-of-the-art optimization techniques are described in Section 4.

2 Preliminaries

A combinational circuit is represented as a Boolean network. A **Boolean network** η is a directed acyclic graph with some primary inputs $PI(\eta)$, primary outputs $PO(\eta)$, and internal (intermediate) nodes $IN(\eta)$. **Primary inputs** have no arcs coming into them, and **primary outputs** have no arcs going out of them. Associated with each **internal node** i of the network is a variable y_i and representation of a logic function f_i. The logic at each node is stored typically as a sum-of-products (SOP) form. There is a (directed) arc from node i to node j in the network if j uses y_i or y_i' explicitly in the representation of f_j. In that case, i is called a **fanin** of j, and j a **fanout** of i. The set of fanins of a node i is denoted as $FI(i)$ and the set of fanouts as $FO(i)$. If there exists a path from node i to node j, then i is said to be a **transitive fanin** of j, and j a **transitive fanout** of i. The set of transitive fanins of a node i is denoted as $TFI(i)$, whereas its transitive fanout set is denoted as $TFO(i)$.

Figure 1 shows a network with four primary inputs a, b, c, and d, three primary outputs x, y, and z, and four internal nodes y, w, x, and z. The primary inputs and output nodes are drawn as squares, and the internal nodes as circles. b and c are fanins of w, and x and z are the fanouts of w. The function associated with w is f_w (also written w) $= bc$. $TFI(z) = \{b, c, d, w\}$. $TFO(w) = \{x, z\}$.

In addition, **arrival time** a_i at each primary input i and **required time** r_j at each primary output j may be given, indicating the times at which signal becomes available at the input i and by which it is required at the output j. These times are derived from the performance constraints on the design: the cycle time and the set-up & hold time constraints

ASP-DAC '97
0-89791-851-7$5.00

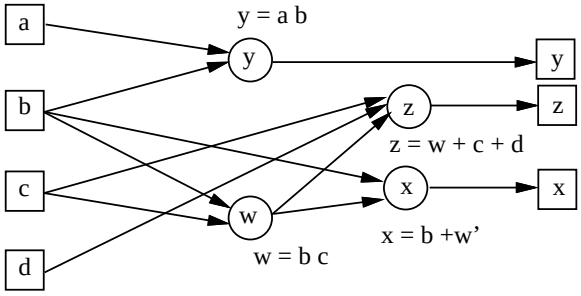


Figure 1: A Boolean network

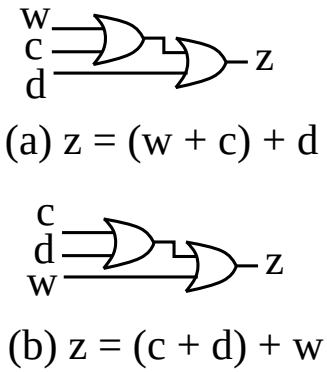


Figure 2: Decomposition of $z = w + c + d$

at the flip-flops and latches.[1]

If arrival time is not specified for some primary input, it is assumed to be zero. Similarly, required times at primary outputs, if unspecified, are forced to be the maximum required time of an output. If no required times are specified, we set the required times of all the outputs to the maximum output arrival time. The arrival times at the primary inputs, required times at the primary outputs, and gate delays are used to compute the arrival time and required time at each node in the network. A **forward delay trace** (from inputs to outputs) of the network determines the arrival time at each node (i.e., at its output). The arrival time at the output of a gate is the sum of the gate delay and the maximum arrival time at its fanins (under a simple gate delay model that ignores effect of the fanout loading, etc). If the computed arrival time a_j at each primary output j is no more than the corresponding required time r_j, the network is said to meet the timing requirements. Similarly, a **backward trace** of the network yields the required time at each node. We compute the **slack** s_i at each node i as the difference between the required time r_i and the arrival time a_i at the node, i.e., $s_i = r_i - a_i$. A node is **critical** if its slack is negative. Critical nodes do not meet the timing constraints and need to be speeded up for improving the circuit performance.

3 Delay Estimation

At technology-independent stage, in the absence of any technology information, delays through network nodes are not known. It is not straightforward to come up with accurate delays of gates and wires, since there is no direct correspondence between Boolean networks and actual circuits based on semi-conductor cell libraries. **Delay models** are used to predict the delay of the nodes and hence through the network. Under **unit delay model**, delay of each gate is one unit. Since the logic function associated with a node may be complex or simple, it is meaningless to assign identical delays to both. So the network is first represented in terms of inverters, 2-input AND gates, and 2-input OR gates. Then it is reasonable to assign unit delay to each such gate. Given a network and a delay model, we can associate delay with each network node (gate). **Delay of a path** is the sum of delays of the gates on the path (ignoring the wiring delays). **Delay through a network** is the maximum path delay in the network. Clearly, this path goes from a primary input to a primary output. For a network represented in terms of two-input gates, the number of **levels** is the same as the delay through the network under the unit delay model.

When decomposing given Boolean functions into two-input gates, we have to be careful about how to decompose, since different decompositions can lead to different delay estimates. For example, node z in Figure 1 realizes a 3-input OR gate. So it needs to be decomposed into two 2-input OR gates. This can be done as shown in either (a) or (b) of Figure 2. Assume that the arrival times of all primary inputs are 0. If we decompose as shown in (a), the delay of z will be 3 using the unit delay model. But if we decompose as in (b), its delay will be 2. We should use the so-called **timing driven decomposition** in order to get more accurate estimation, as will be explained in Section 4.1.1.

Although number of levels is a reasonable estimate of circuit delay, it does not take into account large capacitive loads driven by nodes having many fanouts. Large capacitance can cause large delays. So, an alternative is to use a weighted sum of number of levels and number of fanouts as the delay model. This is called the **unit-fanout delay model** and looks more reasonable intuitively. However, depending on technology mappers or technology libraries, the unit-fanout delay can be less accurate than unit delay.

The above two models, unit delay and unit-fanout delay, along with timing driven decomposition methods, are currently the basic techniques for delay estimation. Appropriateness of these delay models, however, is not clear. For example, we estimated the delays of a set of circuits with both models. Then, we mapped the circuits using two different technology mappers – one was a public-domain mapper and

[1] Almost all designs are sequential in nature – they have memory elements such as flip-flops and latches. Since the problem of sequential circuit optimization is hard, typically only the combinational part between the memory elements is optimized.

the other, an in-house mapper. We obtained mixed results – unit delay model gives better estimation for one mapper and unit-fanout delay model, better for the other mapper. So the results are not conclusive. However, both models give reasonably good correlation with actual delays after technology mapping. That, along with the simplicity of the models, explains why these models are widely used in actual timing optimization tools.

4 Performance Optimization

The performance optimization problem can be stated as follows: *Given an initial circuit function description, a library of primitive gates, and performance constraints, generate an implementation of the circuit using the primitive gates such that performance constraints are met and the circuit area is acceptable.*

Various solutions have been proposed to address this problem. They can be grouped in three broad categories:

1. **Circuit restructuring**: The structure chosen to represent a function determines its delay. Restructuring techniques [7, 19, 17, 3, 5, 21, 4, 16] aim at reducing the delay of the circuit by modifying the structure. For instance, a ripple-carry adder, which is a *serial* implementation of the carry-out, can be restructured to obtain a more *parallel* carry look-ahead structure [9].
2. **Technology mapping**: This refers to selection of gates from the library to implement the logic of the network without changing the global circuit structure. The selection exploits the delay characteristics of the gates for delay reduction. Key contributions in this area were made in [10, 11, 20, 15].
3. **Fanout optimization**: The capacitive load driven by a gate affects the delay through the gate. The fanout optimization problem is how to redistribute this load such that this delay is minimized. This is achieved by building fanout-trees at the output of gates that have high capacitive loading [2, 18, 20]. The fanout-trees consist of inverting and non-inverting buffers. Buffer insertion is typically part of the technology mapping process, since it is only after gate-selection that one knows the load driven by a gate.

In this paper, we will focus only on circuit restructuring. Many restructuring techniques have been proposed in literature. They are mostly based on transformations that designers use to speed up circuits. Techniques such as LSS [6] and SOCRATES [1] are **rule-based**. They use a pre-defined set of local transformations based on design style and the target technology to improve the delay. Others use an **algorithmic approach** to optimize the circuit. For instance, [7, 19] determine a set of nodes whose restructuring improves the overall circuit performance. The choice of these nodes is made so as to obtain the maximum delay improvement with a minimum area increase. Some algorithmic techniques reduce delay by using network don't cares and permissible functions to re-express node functions using early arriving signals [3, 5]; some others use delay-driven clustering and subsequent collapsing and optimization [21], while others either eliminate long paths that cannot be sensitized by any input vector [4] or reduce the delay of the longest sensitizable path [16]. It should be noted that restructuring can be done both before and after technology mapping.

The generic restructuring paradigm can be described as follows.

```
while (circuit timing improves &&
       timing constraints are not met) {
   Select regions of circuit to restructure
   Restructure the selected regions
}
```

Next, we describe some restructuring (or transformation) techniques. We should mention that typically the input network is in terms of 2-input gates. However, the transformations are effective on complex logic functions. Given a two-input gate g, a complex function is obtained by collapsing an appropriate subset of transitive fanin $TFI(g)$ into g. This transitive fanin subset along with g constitutes the **scope of the transformation**, which represents the region to be restructured. Larger the scope, better is the optimization potential. However, since gates in the scope may fanout to other parts of the network, they need to be duplicated. So a larger scope typically results in a higher area penalty. Also, the resulting complex logic function obtained after collapsing may be too large to be represented. State-of-the-art systems, such as speed_up [17], define scope as either the entire transitive fanin up to a depth d or just the critical part. In practice, d is assigned a value in the range 3 to 6.

4.1 Delay Reducing Transformations

We describe some transformations that alter the structure of a part of the circuit such that the delay through the part is reduced.

4.1.1 Tree-height Reduction: Timing Driven Decomposition

Timing driven decomposition decomposes a complex function f into a structure composed of 2-input functions whose delay is minimum. This is done primarily through extraction of divisors that are good for timing (g is a **divisor** of f if f can be written as $f = gh + r$, where h is non-zero). The goodness of a divisor depends on arrival times of its inputs – a good divisor should not have late arriving signals as its inputs, and the area saving it can lead to. The best divisor g is chosen and substituted into f. Then, g and the modified function f are decomposed recursively. If, at some stage, f does not have any divisors, f is necessarily a sum of disjoint-support product terms. Its decomposition into two-input gates is then done as follows. A Huffman-like procedure is used to come up

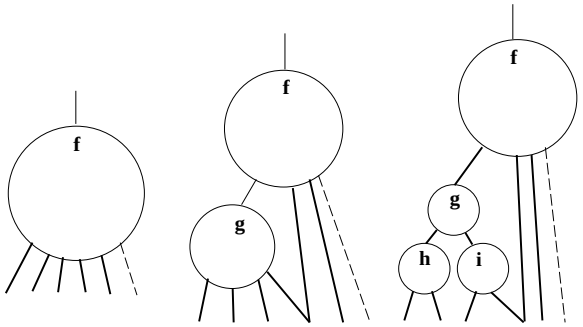
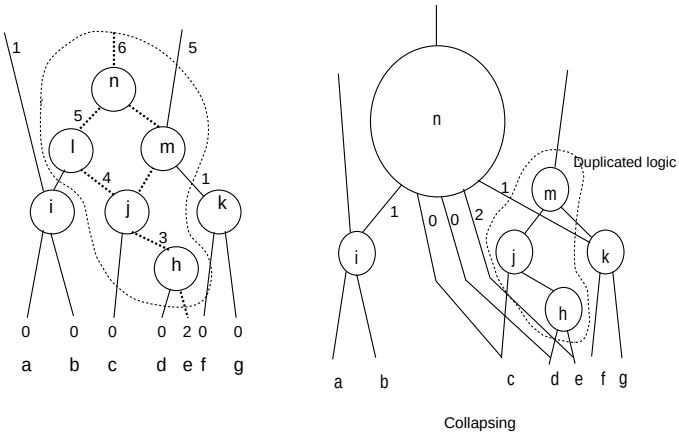


Figure 3: Timing Driven Decomposition



with an optimum 2-input AND decomposition of each product term – at each step, two earliest arriving signals are combined into an AND gate, and are replaced by the AND gate output, whose arrival time is one more than the maximum input arrival time (assuming the unit delay model). This procedure is repeated until only one signal remains [22]. A similar procedure is then used to combine the outputs of the root AND gates realizing the product terms into two-input OR gates.

During the recursive invocation, g is decomposed before f. This is so that when f is decomposed, the arrival time of its input g is known, since it is a 2-input gate. If f were to be decomposed earlier, g could be a complex gate, and we would not know its arrival time. The decomposition process is shown in Figure 3. Dotted lines represent critical signals.

Since the divisors are extracted from an SOP representation of the function f, it can happen that the SOP of the complement of f is better for divisor extraction. It may then be useful to extract divisors from the complement.

Figure 4 illustrates the basic idea behind tree-height reduction. The network shown is in terms of two-input gates. Node n has an arrival time of 6 and is to be speeded up (a unit delay model is assumed). The critical edges and paths are shown with dotted lines. The scope of the transformation is the critical part of $TFI(n)$ and is shown enclosed in the curve. The logic functions in the scope are collapsed into n. Since node m fans out to some node outside the scope, m and its transitive fanin gates in the scope (i.e., j and h) need to be duplicated. Now, n is a complex function and can be redecomposed using timing driven decomposition. This results in a reduction of arrival time from 6 to 4. Note that the decomposition keeps the late arriving input e close to the root node n.

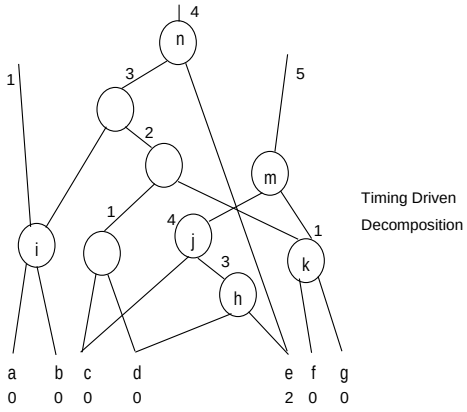


Figure 4: Collapsing and Timing Driven Decomposition

4.1.2 Timing Driven Cofactoring

Given a function f, the latest arriving input x is determined. Then, f is decomposed as $f = xf_x + x'f_{x'}$ [3] (f_x is f with input x set to 1, and $f_{x'}$ is f with x set to 0). A multiplexer is used with its select line tied to x; f_x and $f_{x'}$ are realized, and tied to the 1 and 0 data inputs respectively of the multiplexer. A straightforward implementation realizes f_x and $f_{x'}$ independently, which may result in a large overhead in area. The overhead can be reduced if logic sharing between f_x and $f_{x'}$ is considered. This technique is a generalization of the design of a carry-select adder.

4.1.3 Generalized Bypass Transform

The basic idea in this method is to change the structure of the circuit in such a way that transitions do not propagate along the long paths in the circuit [13]. Given a function f and a late arriving input x, $g = f_x \oplus f_{x'}$ represents conditions under which f depends on x. g is then used as the select signal of the multiplexer, whose output is f. If g is 1, the output f is simply x or x'. If g is 0, the output is the function f with x set to 0 or 1. The long path that depended on x has been replaced by the slower of the two functions: g and f with x set to a constant. This transform is a generalization of the technique used in a carry-bypass adder.

4.1.4 Timing Driven Simplification

Simplification computes a smaller sum-of-products (SOP) representation of a function using a don't care set that is

derived from the network structure and also possibly from the external environment. The goal of timing driven simplification is to compute an SOP that leads to a smaller delay implementation. This is achieved by removing late arriving signals from the current SOP using appropriate don't care minterms, and substituting early arriving signals instead [5].

4.1.5 Clustering and Partial Collapsing

reduce_depth [21] is a technique based on clustering, partial collapsing, and subsequent timing optimization. It is founded on the premise that at technology-independent level, in the absence of technology information and wiring delays, any delay model is going to be inaccurate. So it assigns a zero delay to all the gates, thus treating all input-output paths uniformly. However, whenever a signal crosses cluster boundaries, a delay of one unit is incurred.

First, a clustering is performed using an algorithm proposed in [12], which minimizes the maximum number of times an input-output path has to cross cluster boundaries. The maximum size of a cluster is fixed *a priori*. Each cluster is then collapsed into its root node. The logic within each cluster is then factored and simplified without increasing the number of logic levels. Alternately, one could use transforms proposed earlier, such as timing driven decomposition of Section 4.1.1.

It turns out that in the cluster-generation step, gates can be replicated many times over. So the associated area penalty could be huge. A redundancy removal phase is invoked to recover area. However, this phase can be very time consuming.

4.2 Where to Restructure?

One crucial issue in restructuring is to determine circuit regions that should be transformed. A natural choice is the most critical outputs and their transitive fanins. However, one problem with this approach is that after the most critical outputs have been optimized, outputs that were close to being critical earlier could become critical now (for instance, in Figure 4, after n is optimized, m, with the arrival time of 5, becomes critical). And many useless CPU cycles may be spent optimizing only the most critical outputs – by more than needed amount, and this could also result in an unnecessary area penalty. Thus, it makes sense to optimize close-to-critical nodes along with the most critical nodes. This led researchers to introduce the concept of **ϵ-criticality** [7], according to which a node is deemed ϵ-critical if its slack falls within ϵ of the minimum slack in the network. By selecting an appropriate value of ϵ, the designer can ensure that all ϵ-critical outputs are optimized, potentially leading to better optimization. If ϵ is a user-specified parameter, as is the case in [7, 19], the user can control the outputs to speed up.

The next question is: Given ϵ and the ϵ-critical outputs, where should the transformations described in the last section be applied? One can define the concept of an **ϵ-critical network**, which consists of all the nodes and edges that are ϵ-critical. In order to improve the delays at the ϵ-critical outputs, it is not necessary to speed up the entire transitive fanin; an appropriate separator set of ϵ-critical network would do. A **separator set** (or cut-set) in a network is a set of nodes such that each path from a primary input to a primary output goes through some node in the separator set. A separator set in the ϵ-critical network cuts each ϵ-critical path. If the delay at each node in the separator set is improved (using transformations described in the last section), the entire network speeds up. The idea therefore in [19] was to select a good separator set. Each node in the ϵ-critical network was assigned a weight that represents potential for delay improvement and estimated area penalty if the node along with some transitive fanin logic is restructured. A minimum-weight cutset procedure was used to determine the separator-set. However, there are two problems with this approach:

1. The weight assigned to a node is a combination of area penalty and delay improvement. Since area and delay are measured in different units, they are non-additive and it is not clear how to combine them effectively into one weight number.
2. The delay improvement through a separator set is related to the minimum delay improvement at a node in the set. The weight of a separator set, however, is the sum of the set-node weights, and that includes sum of the delay improvements at the nodes. Thus a minimum-weight cutset, whose weight includes the cumulative delay improvements of the nodes and not the minimum delay improvement, may not be the best way of identifying a choice of nodes to transform from timing viewpoint.

These problems, along with a somewhat arbitrary value of ϵ, prompted Singh [17] to propose

1. an automatic technique for determining ϵ, and
2. a better weighting mechanism.

In the new algorithm, ϵ is determined as follows. First, for each primary output O_i, a lower bound in delay improvement $\delta(i)$ is computed. The network is traversed topologically and for each node being visited, all possible transforms are evaluated and the best transform is selected (the best being determined by the cost function – either the maximum delay improvement or maximum delay improvement per unit area increase). By propagating this information about delay saving at a node, it is possible to compute the guaranteed saving $\delta(i)$ for each output O_i. δ values determine the new slack S that is achievable, and hence $\epsilon = S - s(O)$, O being the most critical output currently. The weight assigned to a node is infinity if its delay improvement is less than ϵ, otherwise it is equal to the area increase as a result of the best transformation. It turns out that to guarantee that the new S is achievable, a minimum-weight separator set may not be sufficient. Instead selection sets have to be enumerated and the best one picked (a **selection set** is simply a set of nodes where transforms should be applied). This is computationally more expensive, but yields good quality solutions. Note that the new weights are only in terms of area penalties, and not delay improvement. Finally, each node on the chosen selector set is restructured using the best transform, as determined earlier.

5 Discussion

We have presented a survey of delay estimation and optimization techniques for combinational logic circuits at technology-independent level. The delay models used today are not very accurate. Furthermore, with the advent of sub-micron technology, wiring delays are becoming increasingly significant contributor to the circuit delay. Since the models currently used do not incorporate wiring delays, inaccuracies are going to get worse. Better delay models that can predict wiring delays are needed. Among various optimization techniques, tree-height reduction seems to be the most powerful and effective transformation. One problem, however, is that state-of-the-art timing optimization tools that use such transformations tend to be really slow. Further research needs to be done to speed up the optimization process itself. Finally, instead of applying transforms at the technology-independent level, one can apply them on the mapped circuit. The hope is that with the delay being more accurately known, these techniques should yield accurate improvements. Unfortunately, preliminary experiments performed in [17] are not encouraging. We, however, believe that this approach deserves a more careful and thorough study, and hopefully significant opportunities for further optimization will be discovered.

References

[1] K. Bartlett, W. Cohen, A. de Geus, and G. Hachtel. Synthesis and Optimization of Multilevel Logic under Timing Constraints. *IEEE Transactions on Computer-Aided Design*, CAD-5(4):582–595, October 1986.

[2] C. L. Berman, J. L. Carter, and K. F. Day. The Fanout Problem: From Theory to Practice. In C. L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, pages 69–99. MIT Press, March 1989.

[3] C. L. Berman, D. J. Hathaway, A. S. LaPaugh, and L. H. Trevillyan. Efficient Techniques for Timing Correction. In *Proceedings of the Internationmal Symposium on Circuits and Systems*, pages 415–419, 1990.

[4] H. Chen and D. Du. Circuit Enhancement by Eliminating Long Paths. In *Proceedings of the Design Automation Conference*, pages 249–252, 1992.

[5] K. C. Chen and S Muroga. Timing Optimization for Multi-Level Combinational Circuits. In *Proceedings of the Design Automation Conference*, pages 339–344, 1990.

[6] J. Darringer, D. Brand, J. Gerbi, W. Joyner, and L.Trevillyan. LSS: A system for Production Logic Synthesis. *IBM Journal of Research and Development*, 28(5):326–328, September 1984.

[7] G. De Micheli. Performance-Oriented Synthesis of Large-Scale Domino CMOS Circuits. *IEEE Transactions on Computer-Aided Design*, CAD-6(5):751–765, 1987.

[8] W. C. Elmore. The Transient Response of Damped Linear Networks with particular regard to Wideband Amplifiers. In *J. Appl. Phys.*, pages 55–63, 1948.

[9] J. P. Fishburn. A Depth-Decreasing Heuristic for Combinational Logic. In *Proceedings of the Design Automation Conference*, pages 361–364, 1990.

[10] K. Keutzer. Dagon: Technology Binding and Local Optimization by DAG Matching. In *Proceedings of the Design Automation Conference*, pages 341–347, 1987.

[11] K. Keutzer and M. Vancura. Timing Optimization in a Logic Synthesis System. In G. Saucier, editor, *Proceedings of International Workshop on Logic and Arch. Synthesis for Silicon Compilers*, pages 1–13, Grenoble, France, May 1988. Inst. Nat. Polytechnique.

[12] E. L. Lawler, K. N. Levitt, and J. Turner. Clustering to Minimize Delay in Digital Networks. In *IEEE Transactions on Computers*, pages 47–57, January 1969.

[13] P. C. McGeer, R. K. Brayton, A. L. Sangiovanni-Vincentelli, and S. K. Sahni. Performance Enhancement through the Generalized Bypass Transform. In *Proceedings of the International Conference on Computer-Aided Design*, pages 184–187. IEEE, 1991.

[14] J. Rubinstein, P. Penfield, and M. A. Horowitz. Signal Delay in RC Tree Networks. In *IEEE Transactions on CAD*, pages 119–127, July 1983.

[15] R. Rudell. Technology Mapping for Delay. part of Ph.D. thesis, March 1989.

[16] A. Saldanha, H. Harkness, P. C. McGeer, R. K. Brayton, and A. Sangiovanni-Vincentelli. Performance Optimization Using Exact Sensitization. In *Proceedings of the Design Automation Conference*, pages 425–429, 1994.

[17] K. J. Singh. *Performance Optimization of Digital Circuits.* PhD thesis, UC Berkeley, December 1992.

[18] K. J. Singh and A Sangiovanni-Vincentelli. A Heuristic Algorithm for the Fanout Problem. In *Proceedings of the Design Automation Conference*, pages 357–360, June 1990.

[19] K. J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Timing Optimization of Combinational Logic. In *Proceedings of the International Conference on Computer-Aided Design*, pages 282–285. IEEE, 1988.

[20] H. Touati. *Performance-oriented Technology Mapping.* PhD thesis, UC Berkeley, November 1990. UCB/ERL M90/109.

[21] H. J. Touati, H. Savoj, and R. K. Brayton. Delay Optimization of Combinational Logic circuits by Clustering and Partial Collapsing. In *Proceedings of the International Conference on Computer-Aided Design*, pages 188–191. IEEE, November 1991.

[22] A. R. Wang. *Algorithms for Multi-level Logic Optimization.* PhD thesis, UC Berkeley, April 1989. UCB/ERL M89/50.