Technology Mapping Using Logical Effort Solving the Load Distribution Problem

Shrirang K. Karandikar and Sachin S. Sapatnekar

Abstract— **Technology mapping is a crucial step in the synthesis of digital designs, and can be used to obtain mapped circuits that are optimized for delay or area. Current tree-based mapping algorithms break the circuit into individual trees and map these optimally. However, these solutions are not globally optimal. This paper presents a new approach to delay-optimal mapping, based on the principle of logical effort. This algorithm maps individual trees such that the solution of the entire circuit is optimal. In traditional technology mapping, the best match for a gate depends on the load being driven, which is not known at the matching stage. Current algorithms handle this situation by generating matches for all loads, and selecting the best match at a later stage. This strategy works for fanout-free circuits, but breaks down at multiple fanout points, where each fanout has to be sized correctly, depending on its criticality. This can have a significant impact on the selection of matches as well, but has not been addressed adequately in the published literature. We refer to the correct sizing of branches of multiple fanout points as the load-distribution problem, which is formally defined and solved in the context of technology mapping in this paper. The effect of the new logical effort based mapping algorithm combined with correct sizing of individual branches of a multiple fanout point leads to implementations that are closer to the global optimum. On average, benchmark circuits mapped using our approach are 37.20% faster and 32.99% smaller than those obtained using SIS.**

I. Introduction

The conversion of a register-transfer level description of a design into an implementation in silicon starts with logic synthesis, which consists of technology independent optimization, followed by technology mapping. In the latter step, the design is mapped to cells belonging to the target library, while optimizing one or more performance metrics, such as delay, area or power. High-performance designs use rich libraries, with multiple instances of each cell, which have various delay, area and drive capabilities. Technology mapping has to not only identify the best logic functionalities of cells to be used to implement some logic, but also the best instance of each selected cell.

This paper addresses the problem of delay-optimal technology mapping, for which a number of algorithms have been proposed in the past, such as tree-mapping [1] and DAG-mapping [2], using load-dependent delay models [3], constant delay models [4], [5] as well as using logical effort [6]. In this paper, the current state of delay-optimal technology mapping is extended in two directions. We present a new approach to mapping, using logical effort (which has previously been used for quick delay estimation and gate sizing). We also present and solve the *load-distribution problem*, which, to the best of our knowledge has not been formally addressed in the literature previously. Optimally solving the load distribution problem can lead to mapped circuits that are more delay- and area-efficient than prior approaches, which have addressed this problem heuristically.

The first contribution of this paper is a new logical-effort based technology mapping algorithm. Logical effort [7], [8] has been widely used in a variety of application domains [5], [9], [10], [11] as well as in industry standard EDA synthesis tools [12], [13]. The method of logical effort primarily provides a quick means of estimating the delay of a path of logic, but has significant drawbacks when analyzing entire circuits having gates with multiple fanouts. This technique also provides a means of calculating the gate sizes that lead to the estimated minimum delay. Consequently, given multiple implementations of the same path of logic, and input and output capacitances for the path, logical effort can be used to easily determine the minimum delay implementation. This notion can be used to constructively map a path of logic to the minimum delay implementation, in a manner similar to traditional mapping algorithms, by enumerating choices and eliminating suboptimal ones. This is the idea behind the new technology mapping algorithms presented in this paper. Our approach, described in Section IV, has a few advantages over previous methods. First, unlike conventional methods, the selection of gate sizes in the solution is implicit, and does not have to be determined during matching. Second, the delay model is inherently load-dependent, and there is no need to enumerate solutions for all possible load values, as is done in the traditional mapping approach [3]. Finally, the size of the library used is much smaller, since each gate need not be instantiated for each available size, leading to faster matches. Combined, these features make our algorithm faster than current algorithms for fanout-free circuits.

The second contribution of this paper is the formulation and solution of the "load distribution problem". Traditional technology mapping approaches partition a circuit into fanout-free trees, and map each tree separately. Every gate within such a tree drives one other gate that is also contained in the tree. The output gate of the tree drives either a primary output, or gates that are input gates of other trees. Within a tree, traditional technology mapping approaches recognize the fact that the delay-optimal match of a gate, and its corresponding size, depend on the load being driven. This is true for the output gate of the tree as well, which drives multiple fanouts. Selecting the correct solution for this output gate is crucial, since this solution, in turn, determines the load for other gates in the tree. However, the load being driven by the output gate of a tree is not known in advance. Additionally, indiscriminately selecting the best solution for each path in a tree can lead to the input gates of the tree having large sizes. While this may lead to each tree in the circuit having the best input-to-output delay, such a solution for the complete circuit is severely sub-optimal. What is required is an approach that takes

This work was supported by SRC under contract 2001-TJ-884 and NSF under award CCR-0205227.

into account the criticality of each component of multiple fanout points, and selects solutions that optimizes the delay of the entire circuit. The problem of assigning correct sizes (or equivalently, capacitances) to gates at multiple fanout points is referred to as the load distribution problem, and is described in greater detail in Section III-B. A solution to this issue in the context of gate sizing has been presented in [14], [15], [16], and is extended to the technology mapping arena in this paper.

II. Background

We first present a brief overview of the method of logical effort, and how it applies to sizing a path for minimum delay. This is followed by a discussion of algorithms that are currently used for technology mapping, and the delay models used in these algorithms.

A. Logical Effort and Gate Sizing

In the method of logical effort, the delay of a gate is estimated by modeling it as a linear function of the load being driven as:

$$\begin{aligned} D &= g \times \frac{c_l}{c_i} + p = g \times h + p \\ &= f + p \end{aligned} \tag{1}$$

where

- **Logical Effort** (g) is the complexity of the gate, relative to an inverter. It measures how much worse the gate is at driving a specified load than an inverter. The base case of an inverter is taken to have unit logical effort, and complex gates such as NAND, NOR and XOR have successively higher values of logical effort. Asymmetric gates, such as AOI or OAI gates have different logical efforts for each input.
- **Electrical Effort, or Gain** ($h = \frac{c_l}{c_i}$) describes how the electrical environment of the logic gate affects performance and how the size of the transistors in the gate determines its load-driving capability. c_l is the load being driven and c_i is the input capacitance of the gate under consideration.
- **Effort Delay** ($f = gh$) is the product of the logical and the electrical efforts of the gate.
- **Parasitic Delay** (p) expresses the intrinsic delay of the gate due to its own internal capacitance, and is largely independent of the size of the transistors in the logic gate.

This formulation separates the different components that contribute to the delay of a gate. More importantly, it leads to a natural extension for estimating the minimum delay, $\hat{D}$, of a *path* of logic as

$$\hat{D} = NF^{1/N} + P \tag{2}$$

where $F = GH$ is referred to as the path effort, P as the path parasitic delay and N is the number of gates on the path under consideration [8]. The path logical effort, G, is the product of the logical efforts of the gates on the path, and the path electrical effort, H, is the product of the gate electrical efforts. The minimum delay of Equation (2) is obtained by distributing the path effort F equally to each gate on the path.

The path electrical effort can also be calculated as the ratio of output and input capacitances of the path. Consider Figure 1,

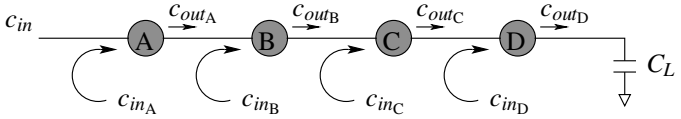


Fig. 1. Calculating the Electrical Effort of a Path

which shows a simple path of four gates, A, B, C and D. Each of these gates have input capacitance c_{in_A}, c_{in_B}, c_{in_C} and c_{in_D}, and drive output capacitances c_{out_A}, c_{out_B}, c_{out_C} and c_{out_D} respectively. The input capacitance of the path, c_{in}, is the input capacitance of gate A, and the output capacitance of the path, C_L, is the output capacitance of gate D. The path electrical effort, H, is the product of the gate electrical efforts, so in this case,

$$\begin{aligned} H &= \frac{c_{out_A}}{c_{in_A}} \times \frac{c_{out_B}}{c_{in_B}} \times \frac{c_{out_C}}{c_{in_C}} \times \frac{c_{out_D}}{c_{in_D}} \\ &= \frac{c_{out_A}}{c_{in}} \times \frac{c_{out_B}}{c_{in_B}} \times \frac{c_{out_C}}{c_{in_C}} \times \frac{C_L}{c_{in_D}} \end{aligned}$$

However, this product telescopes since the input capacitance of each gate is the load capacitance of its input (e.g., $c_{in_C} = c_{out_B}$). Thus,

$$\begin{aligned} H &= \frac{c_{out_A}}{c_{in}} \times \frac{c_{out_B}}{c_{in_B}} \times \frac{c_{out_C}}{c_{in_C}} \times \frac{C_L}{c_{in_D}} \\ &= \frac{C_L}{c_{in}} \end{aligned} \tag{3}$$

The traditional logical effort approach is well suited for estimating the minimum delay that can be achieved by sizing a path of logic (using Equation (2)), if the electrical effort, H, of the path is known. The individual gate sizes that are required to achieve this minimum delay can be calculated as follows. Each gate is assigned a gate effort of $f = F^{\frac{1}{N}}$. Starting with the gate at the output that drives a known load of C_L, the size of each gate is successively determined. Since the logical effort g of a gate is fixed, if an effort delay f is assigned to a gate, the input capacitance c_{in} that meets this effort delay can be calculated as

$$c_{in} = \frac{g \times c_l}{f}, \tag{4}$$

where c_l is the load begin driven by the gate under consideration.

In general, the input capacitance of the k^{th} gate from the output, c_{in_k}, can be calculated as

$$c_{in_k} = \frac{\prod_{i=1}^{k} g_i}{f^k} \cdot C_L \tag{5}$$

Further additions to deal with multiple fanouts in circuits have been presented in [8]. However, as discussed in [16], the "branching effort" has significant drawbacks due to the lack of *a priori* knowledge of the input and load capacitances of individual branches in a circuit. These drawbacks can create inaccuracies when calculating the minimum achievable delay of a circuit, and the corresponding gate sizes, as discussed in greater detail in Section III-B.

B. Traditional Technology Mapping

We now briefly summarize the state of technology mapping and the delay models used in these algorithms. Cell- or library-based technology mapping is the process of binding a technology independent logic level description of a circuit to a library of gates in the target technology. A dynamic-programming algorithm based on tree covering was proposed in [1], and has served

as the basis of later technology mapping algorithms. This is a two-step algorithm:

- In the *matching* step, matches for all gates are generated in an input-to-output traversal of the circuit, and the optimum match (based on its cost and the cost at its inputs), and the corresponding matches at the inputs, is stored as the solution for that gate.
- In the *covering* step, the solution for the entire circuit is generated by an output-to-input traversal of the circuit. At the primary outputs, the best match is selected, and the covering recurses on the inputs of this match.

The delay of a match is a function of the load it is driving, a quantity that is not known during the matching step. In order to account for this in delay-optimal technology mapping, sets of solutions are stored at each gate, each solution being the optimal one for a specific load value. In the covering step, the load is known, and the corresponding optimal match can be selected.

One of the drawbacks of this approach is that the circuit to be mapped (represented by a "subject graph") is partitioned into disjoint fanout-free trees, which are then optimally mapped. However, this leads to restrictions on the solutions, since matches cannot cross tree boundaries. In [2], it was pointed out that if duplication at tree boundaries were to be allowed, DAG-mapping, as opposed to tree-mapping, would provide optimal results. However, this work allocates a fixed, load-independent delay to each gate, and assumes that later gate sizing and buffer tree insertion can achieve the delay assigned. Thus, it does not address the load-distribution problem, described shortly.

The delay models used in technology mapping fall into the following categories:

- **Load-Independent Delay Models** assume that the delay of a cell does not depend on the load being driven, which is unrealistic. However, during technology mapping (even in the case of fanout-free regions), the load is not known until the covering step, and assuming load-independence of delay is convenient. Approaches using these models assume a fixed load during matching, and use buffer insertion and gate sizing after mapping to improve the final delay. This can lead to sub-optimal solutions, when compared to approaches that integrated sizing with mapping.
- **Load-Dependent Delay Models** express the delay as a polynomial function of the load being driven. Higher-order delay functions, such as quadratic functions, can be used for greater accuracy, although linear functions, as used in [3], may also suffice. A technology mapping that uses such a delay function generates optimal matches for all possible load values in the matching step. During covering, the actual value of the load becomes known, and the corresponding match can be selected as the solution.
- **Constant Delay Models** assign a fixed delay to each library cell (note that this is not the same as the load-independent delay models). The technology mapping procedure is carried out under the assumption that given a load, these cells can be sized in order to achieve the assigned delay. These models have been used in [4], [5], but the main drawback is that the optimal selection of cells during matching is sensitive to the load being driven.
- **Gain-Based Delay Models** express the delay of a gate as a function of the ratio of output-to-input capacitance of the gate, and have been applied to technology mapping in [6]. However, the selection of sizes of gates in [6] is based on an ill-defined parameter called *global gain*, whose value is set either by experimentation or relies on the intuition of the designer.

Thus, there have been a variety of approaches to account for the fact that gate delays depend on the load being driven. While technology mapping makes use of these approaches to generate optimal solutions *locally*, the *global* picture is left unfinished i.e., while the selection of the gate types and gate sizes may be optimal within fanout-free trees, the traditional algorithms degenerate into heuristic or greedy approaches at multi-fanout points. This is illustrated in the following section.

III. Drawbacks of Traditional Methods

In traditional tree-mapping methods, the input circuit, represented as a DAG, is partitioned at multi-fanout points into trees, which are then mapped optimally. These algorithms address the fact that size and functionality selection of matches have a significant impact on the quality of the final mapped solution. However, there are a few issues that are not taken into account. First, the selection of the optimal match and the corresponding size is based on the load being driven, and ignores any constraints on the input capacitance of the tree. If this input capacitance is bounded, the best solution may be different from the one selected, as we show in Section III-A. Typically, bounds on the input capacitance are needed, so that the driving gates do not see an unnecessarily large load. The second issue is related – rather than an arbitrary bound on the input capacitances of a tree (which are multi-fanout points in the original circuit), an optimal assignment of capacitances to all fanouts and the driving gate, based on their respective criticalities, can lead to superior solutions, as discussed in Section III-B. Finally, optimally mapping trees need not lead to optimal solutions for the entire circuit, as shown in Section III-C.

A. Load-Dependence of Optimal Matches

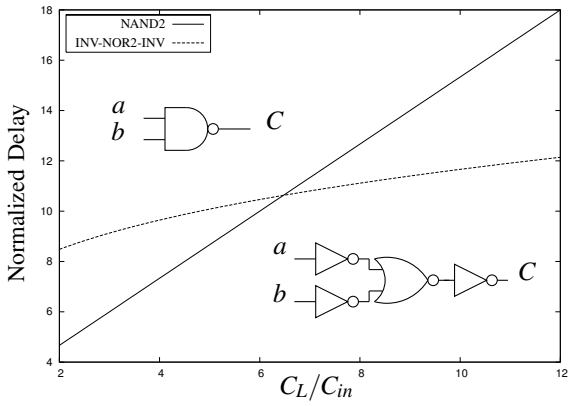


Fig. 2. Influence of Load on Solutions

Consider Figure 2, where output C is the NAND of two inputs, a and b. The load at the output of C is C_L, and the input capacitance is a fixed C_{in}. This functionality can be obtained by either selecting a NAND2 gate directly, as shown on the top, or by selecting an INV-NOR2-INV chain as shown at the

bottom. It may seem that the smaller solution will outperform the larger one. However, consider the delay equations for each option, assuming the following values: $g_{\text{INV}} = 1$, $g_{\text{NAND2}} = \frac{4}{3}$, $g_{\text{NOR2}} = \frac{5}{3}$, $p_{\text{INV}} = 1$ and $p_{\text{NAND2}} = p_{\text{NOR2}} = 2$. The minimum delay that can be achieved by each option can be calculated using Equation (2), where the number of stages, N, is 1 for the NAND2 solution, and 3 for the INV-NOR2-INV solution.

$$\hat{D}_{\text{NAND2}} = \frac{4}{3} \times \frac{C_L}{C_{in}} + 2$$

$$\hat{D}_{\text{INV-NOR2-INV}} = 3 \cdot \left[\frac{5}{3} \times \frac{C_L}{C_{in}} \right]^{\frac{1}{3}} + 4 \quad (6)$$

Figure 2 also plots the minimum delay of Equation (6) as a function of the electrical effort, $\frac{C_L}{C_{in}}$. It is obvious that there is no universally better choice in this case – for small values of electrical effort, the NAND2 has lower delay, while the INV-NOR2-INV is better for larger values of electrical effort. Thus, input *and* output capacitance, rather than the output capacitance only, determine the optimal match, a point that is largely ignored by the traditional technology mapping algorithms. Typically, bigger gates are faster than smaller ones, but have a correspondingly higher input capacitance. In order to ensure that the driving gates at the tree inputs do not see an excessive load, limits on the maximum input capacitance may be enforced. However, these limit are not taken into account when selecting the optimal match.

This example serves to highlight a crucial point – in order to determine optimal matches, we need to know the output as well as input capacitances of logic under consideration. To the best of our knowledge, this aspect of technology mapping has not been taken into account previously. However, it is implicit in our formulation of logical-effort based technology mapping, and in the load-distribution problem and its solution.

B. The Load-Distribution Problem

As seen in the previous section, the optimal solution for a fanout-free tree of logic depends on input as well as output capacitance. The natural question that arises is: in the context of the entire circuit, what are the values of input and output capacitances of the component trees that minimize the delay of the entire circuit? This issue is analyzed in this section.

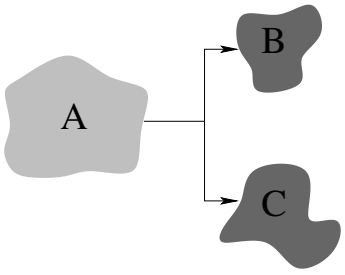


Fig. 3. Assigning Capacitance at Multiple Fanouts for Optimizing Circuit Delay

Consider the situation shown in Figure 3, with a block of logic A fanning out out to two other blocks, B and C, which eventually drive primary outputs. Each of A, B and C are fanout-free regions of the circuit. The optimal solution selected for A depends directly on the load being driven at its output, which, in this case, is the input capacitance of B and C. There are two situations that have to be considered:

- **The interaction between** A **and its outputs** Assigning a larger input capacitance to B and C makes them faster, at the cost of increasing the load on A and slowing it down, and vice versa. What is the optimum value of capacitance that should be assigned to the output of A, so that the delay of the *entire* circuit is minimized?
- **The interaction between** B **and** C The delays of these two fanout-free regions to the primary outputs of the circuit are influenced by their constituent logic and respective input and output capacitances. If the two blocks of logic have very different delays, we would like the critical branch to have a larger input capacitance. On the other hand, if B and C have the same delay, they should have the same input capacitance. Thus, even if we could determine the optimal load that A should be driving, what is the best distribution of this capacitance to each fanout?

We refer to these two problems together as the *load-distribution problem*. Given a load at a multiple fanout point in the circuit, current algorithms can determine the best mapping for the logic up to that point. However, this load is typically estimated using heuristics, and since the mapped solution depends directly on the load being driven, wrong estimates can lead to sub-optimal solutions.

We solve the load-distribution problem by integrating the approach suggested in [16] with technology mapping. This enables us to accurately determine the optimal load that should be driven at a multiple fanout point, and how this load should be distributed, in the form of input capacitance, to each fanout. Once this load has been calculated (as against being estimated), we can use our technology mapping approach to map the circuit.

C. Critical Paths of Trees and of the Entire Circuit

When dealing with fanout-free regions, or trees, in isolation, it is easy to determine the critical input of the tree – this is the tree input that has the maximum delay to the tree output. The path from this critical input to the tree output is the critical path of the tree. Note that the critical path of the mapped circuit may be significantly different from the critical path of the unmapped circuit, depending on the target library. Therefore, the critical path of the mapped tree is not known during the matching phase, and may change depending on the choice of the matches made and the corresponding sizes selected. However, the traditional tree mapping algorithms can map trees so that the delay on the critical path of the mapped tree is minimized.

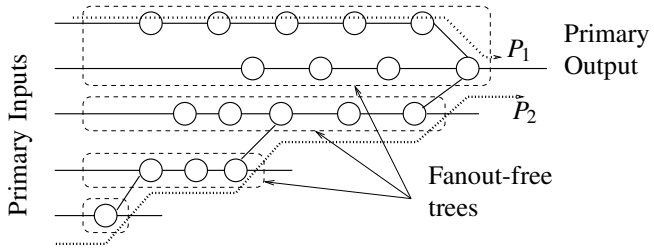


Fig. 4. Critical Path in a Tree and in the Circuit

Now consider the situation when the tree is part of a bigger circuit. In this case, the desired mapped solution is the one that minimizes the delay of the the critical path of the circuit, which may not correspond to the critical path of the constituent trees. This is shown in Figure 4, which shows a part of a circuit that

has been divided into fanout-free trees. Each of these trees fanout to multiple outputs or to primary outputs. In the topmost tree, path P_1 is the path with longest delay. However, the critical path of the circuit is path P_2, which traverses multiple trees. As in the case of trees, the critical path of the unmapped circuit can be very different from the critical path of the mapped circuit, and therefore cannot be used to determine the critical inputs of individual trees.

In such a scenario, optimally mapping individual trees and connecting these mapped trees together can lead to solutions for circuits that are sub-optimal. We address this issue in Section IV-C.3 by generating solutions for each input of a fanout-free segment, and selecting the solution corresponding to the critical input, once it is known, so that the delay through the entire circuit is minimized.

IV. Logical Effort Based Technology Mapping

In this section, we present our approach to mapping a circuit to a target library, that address the drawbacks of traditional approaches presented in the previous section. We first present a logical-effort based technology mapping algorithm for fanout-free circuits. This algorithm is modified in Section IV-B to handle the fact that the critical path in the circuit is not known during the matching step. A key feature of this algorithm is that the solutions generated are functions of input and output capacitances. This allows us to address the load-distribution problem, which uses the concept of Delay-C_{in} curves, and their application to dealing with multi-fanout points, presented in Section IV-C. The combination of our logical effort based technology mapping algorithm and Delay-C_{in} curves leads to our approach for mapping entire circuits optimally, as presented in Section IV-D.

A. Optimal Technology Mapping for Fanout-Free Circuits

We first present our algorithm for a simple path of logic, with each gate having a single input and single output. We then show how this algorithm can be extended to fanout-free circuits, with each gate having multiple inputs but a single output. This algorithm is optimal for trees, but as described in Section III-C, this optimality may not extend to entire circuits. We present a modified version of the matching step, which, when used in conjunction with Delay-C_{in} curves, leads to solutions closer to the global optimum.

In traditional technology mapping, in an input-to-output traversal, all possible matches are generated at each gate. However, only one match (the optimal match, as determined by the cost function), has to be stored for every gate. For minimum-delay technology mapping, the cost function at a gate is the delay of any path from a primary input to the output of the gate, and can be calculated as the sum of the delay of the match itself, and the maximum delay at the inputs of the match. This approach fits the classical dynamic programming paradigm – the delay of the path is optimal only if the delay of a sub-path is optimal; hence all non-optimal matches can be discarded. When gate sizing is taken into consideration, a set of solutions, each corresponding to different possible load values have to be evaluated and stored, rather than a single solution.

Our approach uses the cumulative path logical effort, G as the cost function. Recall that G is the product of all individual gate logical efforts[1] g on any path from a primary input to the gate under consideration. We will show shortly that if the number of gates on a path are taken into account, minimizing G is equivalent to minimizing the delay of that path. This formulation also fits into the optimal substructure property of dynamic programming, since by definition, the minimum value of cumulative logical effort of a path is obtained when the cumulative logical effort of any sub-path is minimum. This formulation has a couple of advantages over previous methods. First, different load values are automatically accounted for, and therefore only one solution has to be stored for each gate. Second, the solution for a path is a function of the input and output capacitances of that path, as shown later in this section. This leads to globally optimal solutions when extended for complete circuits.

We now show how minimizing path logical effort G leads to minimum-delay solutions. Recall Equation (2), which can be expanded as

$$\hat{D} = N(GH)^{1/N} + P \quad (7)$$
$$\text{where } H = \frac{C_L}{c_{in}}$$

Given a path having N stages of logic and path logical effort G and path electrical effort H, the minimum delay in Equation (7) can be obtained by sizing gates on the path appropriately. If we were to have the freedom of replacing gates on the path with functionally equivalent choices, while maintaining the path length and electrical effort, it is obvious that the optimal solution after gate sizing is obtained when the path logical effort G is minimized. Thus, in an input-to-output traversal of a path, selecting the match that minimizes the cumulative logical effort for a certain path length will lead to a solution that minimizes the delay of the mapped path after sizing. For greater accuracy, we use the parasitic delay P as a secondary criteria, to break ties when we have solutions with equal path logical effort.

Of course, the length of the path can vary depending on the matches selected. At each gate, we store a set of optimal matches, for different path lengths. Correspondingly, at the primary output, we obtain a set of solutions of different path lengths and different values of cumulative logical effort. We can then use Equation (7) to determine the combination of path length N, cumulative logical effort G and parasitic delay P that will minimize the delay of the mapped circuit after sizing.

Our logical effort based approach to technology mapping, for a simple path, with a known input and output capacitance can be summarized as follows.

- In the matching step, traverse the path from the primary input to the primary output. For each match at a gate, the cost function is computed as the product of the logical effort of the match and the cumulative logical effort at the input of the match. The length of the path is the length of the input of the match plus 1. For all path lengths, store the best match.
- At the primary output, determine the combination of G, P and N that will minimize the delay after sizing, as calculated by Equation (7).

[1]In this context, "gate logical effort" refers to the logical effort of individual matches.

- In the covering step, traverse the path from the primary output to the primary input, generating the solution as in regular technology mapping. In addition, calculate the correct sizes of each gate using Equation (5).

The above description was restricted to simple paths of logic where each gate has a single fanin and a single fanout. We can now generalize this approach to circuits with gates having multiple fanins (the case with multiple fanouts is handled in the following sections). Since each gate has a single fanout, there is a single path from a primary input to any gate in this circuit. In traditional technology mapping, for some gate t, the input to a match at t with the maximum delay from a primary input is defined as the critical input, and this delay is used in combination with the delay of the match to determine the delay (and hence the cost) at the output of the match. In our approach, the minimum delay of a path is achieved by minimizing the cumulative logical effort for a selected path length. In this case, we determine the critical input and calculate the cost function as follows.

Let the match at gate t have r inputs $I_1, I_2, \ldots I_r$. Consider the situation where this match has some input capacitance c_{in_t}, and the path length from a primary input to any input of the match at t is the same. The input capacitance of the match is the load capacitance that each of $I_1, I_2, \ldots I_r$ have to drive. If the input capacitance at each primary input were also fixed (say equal to c_{inPI}), the electrical effort of all the paths from primary inputs to the input of gate t will be $H = \frac{c_{in_t}}{c_{inPI}}$, and will be equal. Thus, on the basis of Equation (7), the critical input I_c, which has maximum delay from its primary input, is the one that has the maximum cumulative logical effort for that path length. When calculating the cost of a match at gate t for some value of path length N, we need to determine the cumulative logical efforts at each input of this match for path lengths $N-1$, and the maximum of these is used to calculate the cumulative logical effort at the output of t.

The above argument makes a couple of assumptions that may seem restrictive. However, we will show that these do not affect the definition of the critical input. The first assumption is that a match will present the same input capacitance on every input pin. This is true for symmetric gates (such as NANDs or NORs), but not for asymmetric gates, such as AOIs or OAIs. However, we shown in Lemma 1 that even with each input to a match having different electrical efforts, the critical input is still the one with maximum cumulative logical effort, as defined above. The other assumption is that every input has solutions corresponding to a given path length. That is, when determining the optimal solution for path length n, our approach assumes that solutions of path length $n-1$ are available at each input of the match. This may not be the case in general, and remains a drawback of our approach. However, even with this drawback, the solutions obtained by our approach are significantly better than the ones obtained by traditional methods, as will be seen in the results presented in Section V.

A final consideration that has to be accounted for is the load seen by non-critical inputs of a match after sizing the final mapped circuit. The cumulative logical effort up to a gate being mapped is calculated by taking the product of the logical effort of the match itself and the cumulative logical effort of the critical input of the match. Tracing the path from the primary output to a primary input, following the critical input at each gate defines the critical path in the circuit under consideration. At the primary output, the cumulative logical effort is used to size this critical path, as shown in Section II-A, so that the delay on this path is minimized. The non-critical inputs of gates on this path, however, have no choice in this sizing. Is it possible that the load seen by non-critical inputs becomes large enough, so as to make them critical? As the following Lemma shows, this is not the case.

Lemma 1: Let I_c be the critical input of a gate t, as defined previously. After sizing t and its outputs, I_c is still the critical input of t.

Proof: See Appendix.

Our logical effort based technology mapping procedure for fanout-free circuits can be carried out in a manner similar to the approach for simple paths. The optimum solution at each gate is determined for all values of path lengths. For some path length N under consideration, the cost of each match is the product of the logical effort of the match, and the maximum of the costs at its inputs, corresponding to lengths $N-1$. At the primary output, for some electrical effort, the combination of path length, cumulative logical effort and parasitic delay that minimizes the delay of the circuit as determined by Equation (7) is selected (this assumes that gate sizing will be applied to the selected solution). If only such a fanout-free circuit is to be mapped, the electrical effort is known. If this fanout-free circuit is part of a bigger circuit, the electrical effort is determined using Delay-C_{in} curves, discussed in the following sections.

The pseudo-code of our dynamic-programming based algorithm for technology mapping fanout-free circuits is presented in Algorithm 1.

An Illustrative Example: We use Figure 5 to illustrate Algorithm 1. Here, a simple chain of three gates, A, B and C is to be mapped to a library of three cells, X, Y and Z, with logical efforts g_{X}, g_{Y} and g_{Z} and parasitic delays p_{X}, p_{Y} and p_{Z}.

As discussed before, we store optimal solutions for each legal value of path length. For each gate t we keep track of the accumulated product of logical efforts G_t, the sum of the parasitic delays P_t, and the corresponding matches $\mathcal{M}_t$, indexed according to the length of the path. The path length is obtained by the length at the inputs to the match at t plus one for the match itself[2].

In the example, the only match of a library pattern at gate A is that of pattern X, and the corresponding solution for A is $G_{\mathrm{A}}[1] = g_x$, $P_{\mathrm{A}}[1] = p_x$, $\mathcal{M}_{\mathrm{A}}[1] = \mathrm{X}$. At gate B, however, we have two possible matches, the match of X, with solution $G_{\mathrm{B}}[2] = g_x^2$, $P_{\mathrm{B}}[2] = p_x + p_x$, $\mathcal{M}_{\mathrm{B}}[2] = \mathrm{X}$, and the match of Y, with solution $G_{\mathrm{B}}[1] = g_y$, $P_{\mathrm{B}}[1] = p_y$, $\mathcal{M}_{\mathrm{B}}[1] = \mathrm{Y}$. Thus, B has two solutions of length 1 and 2. At gate C, all three library patterns match, generating the following solutions:

- Match of Z: This is straightforward, with the solution being $G_{\mathrm{C}}[1] = g_z$, $P_{\mathrm{C}}[1] = p_z$, $\mathcal{M}_{\mathrm{C}}[1] = \mathrm{Z}$.
- Match of Y: In this case, the input to the match is A, and the only input solution available is of length 1. Hence, the corresponding solution for C, of length 2, is $G_{\mathrm{C}}[2] = g_x \cdot g_y$, $P_{\mathrm{C}}[2] = p_x + p_y$, $\mathcal{M}_{\mathrm{C}}[2] = \mathrm{Y}$.

[2]For example, $G_t[3]$ is the cumulative logical effort of a path having length 3, and the corresponding match is stored in $\mathcal{M}_t[3]$

Algorithm 1 Optimal LE-Based Technology Mapping for Fanout-Free Regions

// G_t is the cumulative logical effort at the output of gate t, indexed by path lengths
// $\mathcal{M}_t$ is the set of selected matches at t, indexed by path length
// $\mathcal{M}$ is the set of all possible matches at gate t
// I is the set of inputs to match m

// initialize
for each primary input p **do**
  $G_p[0] = 1$
end for

// Phase I: Matching
for each gate t in topological order **do**
  set $\mathcal{M}_t[n] = 0$ for all n
  for each $m \in \mathcal{M}$, with logical effort g_m **do**
    for each available path length n **do**
      // calculate cumulative effort $G_t[n]$ from the inputs,
      // using solutions corresponding to path length $n-1$
      $temp = g_m \times max_{i \in I} G_i[n-1]$
      if $\mathcal{M}_t[n] = 0$ OR $temp < G_t[n]$ **then**
        $G_t[n] = temp$
        $P_t[n] = p_m + P_i[n]$
        $\mathcal{M}_t[n] = m$
      end if
    end for
  end for
end for

// Phase II: Selecting Solution
at the primary output, select the combination of G, H and N that minimizes delay

// Phase III: Covering
select matches in a traversal from the primary output to primary inputs, sizing the matches appropriately

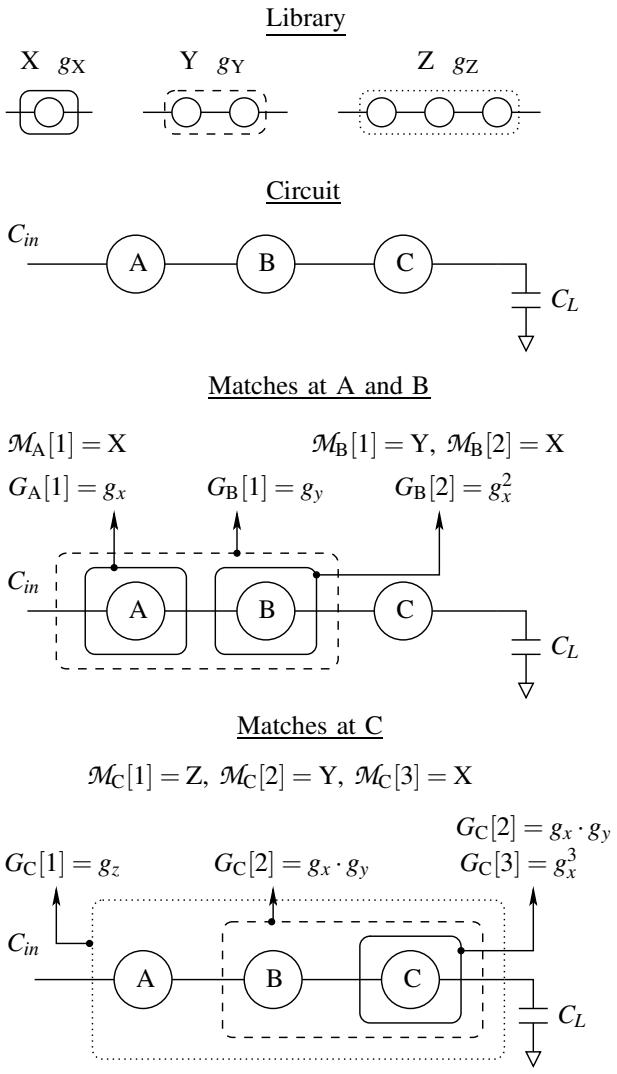


Fig. 5. Example of LE-Based Technology Mapping

- Match of X: The input to this match is B, which has two solutions. Each of these leads to two solutions for C, of length 2: $G_C[2] = g_y \cdot g_x$, $P_C[2] = p_y + p_x$, $\mathcal{M}_C[2] = X$ and of length 3: $G_C[3] = g_x^3$, $P_C[3] = 3 \cdot p_x$, $\mathcal{M}_C[3] = X$.

Note that we now have two solutions at circuit node C of length 2, due to the matches of Y and X. We store the solution with the minimum value of cumulative logical effort.

As we have reached the primary output, the matching step is complete. We have three possible solutions at the primary output, of lengths 1, 2 and 3. The load C_L is known for each solution, and assume that the primary input has a fixed drive capability of C_{in}. This determines the electrical effort $H = \frac{C_L}{C_{in}}$, and we can calculate the minimum delay corresponding to each available solution using Equation (7), and select the minimum. In this case, we have the minimum delay of the mapped circuit with path length 1 to be

$$\begin{aligned} \hat{D} &= 1 \cdot (G_C[1] \cdot H)^{1/1} + P_C[1] \\ &= g_z \cdot H + p_z, \end{aligned}$$

for path length 2

$$\begin{aligned} \hat{D} &= 2 \cdot (G_C[2] \cdot H)^{1/2} + P_C[2] \\ &= 2 \cdot (g_y \cdot g_x \cdot H)^{1/2} + p_y + p_x, \end{aligned}$$

and for path length 3

$$\begin{aligned} \hat{D} &= 3 \cdot (G_C[3] \cdot H)^{1/3} + P_C[3] \\ &= 3 \cdot (g_x^3 \cdot H)^{1/3} + 3 \cdot p_x, \end{aligned}$$

As discussed before, the individual gate sizes can then be determined using Equation (5).

As described in Section II-B, traditional approaches calculate and store solutions for all possible load values. We trade this off with generating solutions for different values of path length, N. The number of legal values of N depend on the circuit being mapped, but also depends on the library. During the mapping stage of Algorithm 1, complex gates, if available in the library, will cover a greater part of the circuit while increasing path length by 1. In comparison, simple gates will increase the path length by a larger amount. We also note that the number of path lengths is not the same for all gates in a path, but increases as we traverse the path from the input to the output. Thus, in practice, we find that keeping track of N solutions at each gate is faster than keeping track of solutions for each load value, as done in

the traditional approach.

If we map fanout-free regions only, Algorithm 1 provides optimal solutions. The path effort F can be used to calculate the sizes of each match selected on the critical path. Matches that are not on the critical path can be sized once the critical path has been fixed. The non-critical paths have a certain amount of slack in the delay that they have to meet. This slack, and the fact that the critical path is now presenting a load that is smaller than previously anticipated can be used in a possible optimization, to control the area of the implementation.

B. Generalized Matching for Fanout-Free Regions

As presented in Section III-C, the critical path of a circuit may not correspond to the critical path of a fanout-free region or tree. Therefore, while Algorithm 1 can optimally map fanout-free regions, the optimal solution for a tree may not provide the minimum delay for the entire circuit. This drawback also exists in current tree-mapping algorithms. The main problem with correctly addressing this issue is that the critical input (for minimizing circuit delay) of a fanout-free region is not known during the matching phase, and can change depending on the selections made (and their corresponding size assignment) during technology mapping. In this section, we modify the matching step of Algorithm 1 so as to obtain generalized solutions for fanout-free trees. In the covering step, these solutions can be used to select the true critical input, and the corresponding optimal matches.

Consider a fanout-free region having inputs $s_1, s_2, \ldots, s_k$. In a tree, each gate has only one output, and therefore if there is a path from an input s_j to a gate t in the fanout-free region, this path is unique. In Algorithm 1, we stored one solution for each path length for gate t, with cost $G_t[n]$, and the optimality of this solution was based on determining the critical input to the match. In the context of the entire circuit, we can no longer use the critical input within a fanout-free region to determine global optimality. Instead, we store k solutions for each path length, corresponding to each input s_j of the fanout-free region, denoted by $G_{s_j \to t}[n]$.

The modified version of the matching step is shown in Algorithm 2. Consider the situation when matching at some gate t. Each input i of the match has a set of solutions of the form $G_{s_j \to i}[n]$, where there exists a path from the j^{th} input of the tree to i. This can be combined with the current match to obtain the solution for gate t, $G_{s_j \to t}[n+1]$. Other matches for gate t of path length $n+1$ from tree input s_j can produce different costs, and as before, we store the minimum cost solution. In this manner, we track solutions of different path lengths from each tree input, and defer the determination of criticality to a later stage. The complexity of doing so increases by $O|\text{FI}|$, where $|\text{FI}|$ is the number of tree inputs, which can be potentially large. However, we show in Section V that this number is less than three on average. Also note that the number of tree inputs that fanin to a gate increases with the depth of the tree, and it is only the output of the tree that has to keep track of solutions from each tree input.

Algorithm 2 is an algorithm for the matching step for fanout-free circuits, which generates sets of solutions for the fanout-free region corresponding to different inputs of the region and different path lengths. Given an electrical effort (output load capacitance and input capacitance), the optimal matching solution is readily determined. The covering step is based on Delay-C_{in} curves, and is described in the following sections. Determining which input is critical is handled after covering.

Algorithm 2 Optimal LE-Based Matching for Fanout-Free Regions

// The fanout-free region has k inputs, s_1, s_2,…,s_k
// $G_{s_j \to t}$ is the cumulative logical effort from input s_j to the output of gate t, indexed by path lengths
// $\mathcal{M}_{s_j \to t}$ is the set of selected matches at t for each input s_j, indexed by path length
// $\mathcal{M}$ is the set of all possible matches at gate t

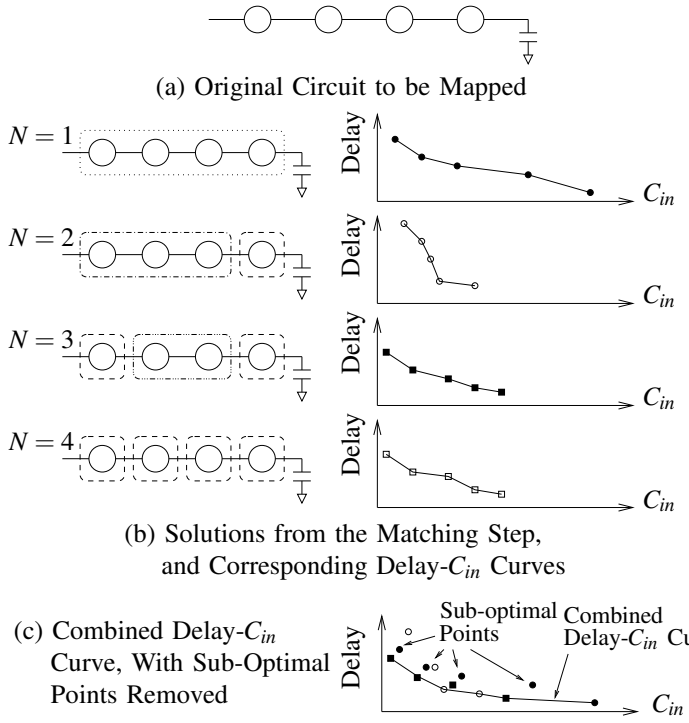
// initialize
for each input s_j **do**
  $G_{s_j \to s_j}[0] = 1$
end for

for each gate t in topological order **do**
  set $\mathcal{M}_{s_j \to t}[n] = 0$ for all inputs s_j and path lengths n
  for each $m \in \mathcal{M}$ **do**
    for each available path length n **do**
      // calculate cumulative effort $G_{s_j \to t}[n]$ from the inputs of the match,
      // using solutions corresponding to path length $n-1$
      for each input i of match m, having logical effort g_{m_i} **do**
        for each input s_j of the fanout-free region having a path to i **do**
          $temp = g_{m_i} \times G_{s_j \to i}[n-1]$
          if $\mathcal{M}_{s_j \to t}[n] = 0$ OR $temp < G_{s_j \to t}[n]$ **then**
            $G_{s_j \to t}[n] = temp$
            $P_{s_j \to t}[n] = p_m + P_{s_j \to i}[n]$
            $\mathcal{M}_{s_j \to t}[n] = m$
          end if
        end for
      end for
    end for
  end for
end for

C. Delay-C_{in} Curves and their Efficient Calculation

We now turn to the case of gates having multiple fanouts. Here, the correct choice when mapping and sizing each branch depends on which one is critical, as formalized by the load distribution problem. Traditional approaches handle each branch separately, but it is clear that this can lead to suboptimal solutions. Our approach to a globally optimal solution uses the notion of Delay-C_{in} curves, previously developed for gate sizing [16]. We show how Delay-C_{in} curves can be calculated easily when integrated with the technology mapping algorithm presented in Section IV-A.

The Delay-C_{in} curve is characterized at the input of each fanout-free segment of the circuit. Each point on the curve corresponds to the minimum delay of the critical path from that input to some primary output, for different values of input capacitance. Which primary output terminates the critical path is



(a) Original Circuit to be Mapped

(b) Solutions from the Matching Step, and Corresponding Delay-C_{in} Curves

(c) Combined Delay-C_{in} Curve, With Sub-Optimal Points Removed

Fig. 6. Delay-C_{in} Curves Calculation for a Tree Driving a Primary Output

immaterial, in fact is is possible that different paths are critical for different values of input capacitance. For some input s, the minimum delay on its Delay-C_{in} curve is represented by $D_{s\rightarrow PO}[c_{in}]$. The critical path may traverse multiple trees, each of which have multiple fanouts. As shown in Section III-B, correctly assigning capacitance to each fanout can have a large affect on circuit performance. Delay-C_{in} curves keep track of this information in addition to the minimum delay values for different input capacitances.

We calculate Delay-C_{in} curves for each input of every tree in the circuit in a recursive manner, starting from the primary outputs and traversing trees in reverse topological order to the primary inputs. There are three main situations that have to be handled, the base case of the primary outputs, trees that only drive primary outputs, and finally trees that have multiple fanouts, as follows.

1) Primary Outputs: The Delay-C_{in} curve at a primary output has only one point – a delay of zero for the fixed load being driven. If a required arrival time is specified for each primary output, a proportional delay value can be used in the Delay-C_{in} curve. For example, the primary output with the smallest required arrival time is assigned a delay of zero in its Delay-C_{in} curve, and the other primary outputs are assigned delays equal to the difference in their required arrival times and the smallest required arrival time. In addition, if the circuit being mapped is part of a bigger design, the effects of different load capacitances at the primary outputs can be captured by adding these to the Delay-C_{in} curve of the primary output, thus allowing for an exploration of a much larger space of solutions. This does not add to the complexity of our algorithm.

2) Trees Driving Primary Outputs: When the fanout-free region drives a primary output, the load being driven is known and is fixed, and the Delay-C_{in} curve is straightforward to calculate. Consider Figure 6(a), where the circuit drives a fixed load of C_L at a primary output. Our matching algorithm generates four possible solutions, of lengths one to four. For each solution, we know the minimum delay can be obtained by the formula

$$\begin{aligned}\hat{D} &= N(GH)^{1/N} + P \\ &= N(G \times \frac{C_L}{c_{in}})^{1/N} + P,\end{aligned}$$

where the matching step gives different values of cumulative logical effort, G, for each value of path length, N. The minimum delays for each solution are therefore functions of the input capacitance, c_{in} -

$$\begin{aligned}\hat{D} &= 1 \cdot (G[1] \cdot \tfrac{C_L}{c_{in}})^{1/1} + P[1] \text{ for } N = 1 \\ \hat{D} &= 2 \cdot (G[2] \cdot \tfrac{C_L}{c_{in}})^{1/2} + P[2] \text{ for } N = 2 \\ \hat{D} &= 3 \cdot (G[3] \cdot \tfrac{C_L}{c_{in}})^{1/3} + P[3] \text{ for } N = 3 \\ \hat{D} &= 4 \cdot (G[4] \cdot \tfrac{C_L}{c_{in}})^{1/4} + P[4] \text{ for } N = 4\end{aligned}$$

Different values of c_{in} give us different delays to the primary output for each of the above possibilities. These constitute the Delay-C_{in} curves for each of four mapped solutions, as shown in Figure 6(b). It is not necessary to keep track of each of these curves, instead, they can be combined to obtain the curve shown in Figure 6(c), by selecting the minimum delay value for each possible c_{in}. Points on this plot that do not lie on the Delay-C_{in} curve are suboptimal, and can be disregarded, since they represent solutions that have higher input capacitance and greater delay than the points on the curve. Thus, in the context of technology mapping, the Delay-C_{in} curve also keeps track of which path length each point on the curve corresponds to.

Note that by calculating the Delay-C_{in} curve, we have still not selected any particular match as optimal at this stage. After the matching step, solutions were generated for different path lengths, and the electrical effort was not known. After calculating Delay-C_{in} curves, the dependency on path lengths is removed, since each point on the curve explicitly corresponds to the best value of path length for that input capacitance. In addition, the set of solutions is now a function of input capacitance – once this is known, the optimal match is also known. In the current case of the tree driving a primary output, the load capacitance is fixed, however, in the general case, presented next, the optimal load capacitance is also determined when calculating the Delay-C_{in} curves.

The example in Figure 6 shows a single input fanout-free region. In general, Delay-C_{in} curves can be calculated for every input of a tree with multiple inputs. In this calculation, the implicit assumption is that the input being considered is the critical input. Whether this is really the case is not known until the covering step, when the appropriate solution is selected.

3) Trees With Multiple Fanouts: The final case that has to be taken into consideration when calculating Delay-C_{in} curve is the most general one, that of an intermediate tree, such as the one shown in Figure 7, where a fanout-free region with output t and inputs $s_1, s_2, \ldots, s_k$ drives multiple fanouts, $F_1, F_2, \ldots, F_l$, each of which may be inputs of other fanout-free regions or may be primary outputs. The load that t has to drive, C_L is the sum of the input capacitances of each of its fanouts, $C_L = \sum_{j=1}^{l} c_{in_{F_j}}$.

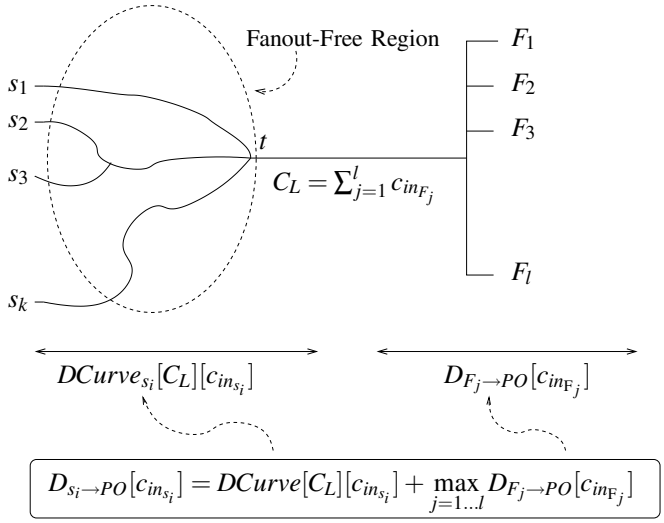


Fig. 7. Delay-C_{in} Curves and Multiple Outputs

Since this is a fanout-free region, there is exactly one path from each of the s_i to output t, and after the matching step, we have solutions for each input of the fanout-free region, for different path lengths. For some load C_L at the output of t, we can calculate the minimum delay from s_i to t for different values of $c_{in_{s_i}}$, as done in the previous section. This is denoted by $DCurve_{s_i}[C_L][c_{in_{s_i}}]$ and is one component of the critical path delay from s_i to a primary output. The second component is the delay from the output of t (or equivalently, the input of the critical fanout, F_j) to a primary output. In order to determine which fanout is critical, we need to know the minimum delays from each fanout to any primary output. However, this information is readily available in the Delay-C_{in} curves of the fanouts – recall that this is denoted by $D_{F_j \to PO}[c_{in_{F_j}}]$. In order to calculate the Delay-C_{in} curve at s_i, the above sum (of $DCurve[C_L][c_{in_{s_i}}]$ and $\max_{j=1...l} D_{F_j \to PO}[c_{in_{F_j}}]$ has to be repeated for all values of C_L, and the minimum is taken. Thus,

$$D_{s_i \to PO}[c_{in_{s_i}}] = \min_{C_L} \left\{ DCurve[C_L][c_{in_{s_i}}] + \max_{j=1...l} D_{F_j \to PO}[c_{in_{F_j}}] \right\} \quad (8)$$

Algorithm 3 shows how the Delay-C_{in} curve of input s_i of a fanout-free region terminating in t can be calculated. Given an electrical effort, $H = \frac{C_L}{c_{in_{s_i}}}$, the first procedure, Calculate $DCurve_{s_i}$ is used to calculate the best delay of the fanout-free region from all the solutions of different lengths that have been generated by Algorithm 2. Given the electrical effort $H = \frac{C_L}{c_{in}}$, this procedure determines the length of the path and the cumulative logical effort that supply the best delay. Procedure Calculate $D_{s_i \to PO}$ of Algorithm 3 determines the best load, and the best distribution of this load to all fanouts, for the given input capacitance, as described above.

While it may seem that the total number of combinations of $c_{in_{F_j}}$ is large (it is in fact $O(|c_{in_{F1}}| \times |c_{in_{F2}}| \times \ldots |c_{in_{Fl}}|)$, where $|c_{in_{F_j}}|$ is the number of possible values of input capacitance of F_j), it is shown in [16] that the number of combinations that actually have to be considered is much smaller ($O(|c_{in_{F1}}| + |c_{in_{F2}}| + \ldots |c_{in_{Fl}}|)$). This is accomplished as follows. We take the minimum values of $c_{in_{F_j}}$ for all j as the first combination, and

Algorithm 3 Calculating the Delay-C_{in} Curve for Input s_i of a Multiple-Fanout Tree

```
Calculate DCurve_{s_i}[C_L][c_{in_{s_i}}]
// s_i is the input, t is the output of the path
for all values of path length n do
    temp = n [ G_{s_i→t}[n] × C_L / c_{in_{s_i}} ]^{1/n} + P_{s_i→t}[n]
    if temp < DCurve_{s_i}[C_L][c_{in_{s_i}}] then
        DCurve_{s_i}[C_L][c_{in_{s_i}}] = temp
        Store n
    end if
end for

Calculate D_{s_i→PO}[c_{in_{s_i}}]
// t has l outputs, F_0, F_1 ... F_l
for every combination of c_{in_j} of all fanouts F_j do
    if the selected combination is not redundant then
        C_L = Σ_{j=1}^{l} c_{in_j}
        Calculate DCurve_{s_i}[C_L][c_{in_{s_i}}]
        temp = DCurve[C_L][c_{in_{s_i}}] + max_{j=1...l} D_{F_j→PO}[c_{in_{F_j}}]
        if temp < D_{s_i→PO}[c_{in_{s_i}}] then
            D_{s_i→PO}[c_{in_{s_i}}] = temp
            Store the input capacitances c_{in_{F_j}} of each fanouts
        end if
    end if
end for
```

sort the fanouts by delay. Say F_1 is the critical fanout in this case. All combinations of $c_{in_{F_1}}$ other than the current one can be ignored, since they will lead to a higher value of C_L and have the same maximum delay to a primary output. The next combination can be obtained by selecting the next value of $c_{in_{F_1}}$, and again determining the most critical fanout.

As discussed in the previous section, in the matching step (Algorithm 2), for this fanout-free region, we had obtained a set of solutions for different values of path lengths N for input s_i, and the electrical effort (the ratio of the output and input capacitances) was an unknown. After the Delay-C_{in} curve has been calculated, we now have a solution for each value of input capacitance, and the dependence on path length has been removed. The unknown value of output capacitance, C_L, that gives us the best delay is now a known quantity, and is embedded in the Delay-C_{in} curve.

Recall the load distribution problem at a gate with multiple fanouts, in which the correct assignment of capacitances between a gate and its fanouts, and the correct assignment of capacitances between the fanouts could have a large impact on the overall delay of the circuit. In Algorithm 3, we consider all values of load capacitance C_L when selecting the optimal solution for input s_i of the fanout-free region. This handles the first part of the load distribution problem, that of the correct distribution of capacitance between a gate and its multiple fanouts. The different values of load capacitance, C_L are obtained by considering all combinations of input capacitances of the fanouts, $c_{in_{F_j}}$. This implicitly handles the second aspect of the load distribution problem, that of distributing a capacitance between multiple fanouts. For some distributions, a particular fanout F_x may be

critical, for some other, another fanout F_y may be critical. This is taken care of by the formulation of Equation (8) and in Algorithm 3.

Algorithm 3 is a dynamic programming algorithm. The Delay-C_{in} curves of one fanout-free region are calculated based on the curves at its outputs, and a particular critical path delay is obtained by simply taking the combination of the delay of the fanout-free region with the maximum critical path delay of the outputs. This also exhibits optimal substructure, since the delay of the critical path is minimized only when the delay of each component of the path is minimized. Hence the delay curves obtained at primary input encode globally optimal solutions to the load-distribution problem.

D. A Comprehensive Technology Mapping Approach

The complete approach for logical effort based technology mapping addressing the load-distribution problem, called MELT (Technology Mapping using Logical Effort: the order of letters are suggestive of the multiple input-output-input traversals of the circuit required by our approach) is presented in Algorithm 4. After the first three steps, which have been described previously, we have Delay-C_{in} curves at the primary inputs of the circuit. At each primary input, the load that minimizes the maximum delay to any output is selected. The primary inputs are processed in decreasing order of this delay. A forward traversal from the primary inputs using the selected loads fixes the input and output capacitances and the lengths of each fanout-free region. This information, in turn can be used to select the matches of the optimal solution.

Algorithm 4 MELT: Technology Mapping using Logical Effort

Divide the circuit into fanout-free regions

PI→ PO Traversal: generate matches for each fanout-free region using Algorithm 2, storing optimal matches for each input of the fanout-free region

PO→ PI Traversal: calculate Delay-C_{in} curves for each input to the fanout-free region using Algorithm 3

PI→ PO Traversal: select the optimal electrical effort for each fanout-free region, and the corresponding lengths

Covering: use the assigned output and input capacitances to generate the corresponding optimal covers for each fanout-free region

In Algorithm 4, there are two issues that restrict the optimality of the final solution. First, the processing of each input of a fanout-free region is carried out independent of other inputs of this region. The solutions generated by different inputs may contradict each other. Second, in general, circuits have reconvergent fanouts. The interaction between multiple, overlapping reconvergent paths is difficult to analyze efficiently. For both these cases, we use the heuristic of assuming that all paths are independent, and make the best choice available. The loss of optimality is acceptable when compared with the alternative of calculating the exact solution.

The first step in Algorithm 4, that of generating matches, takes time $O(|V|+|E|)\cdot|L|\cdot|N|\cdot|FI|$ for each tree, where $|V|$ is the number of nodes and $|E|$ is the number of edges in the tree, $|L|$ is the size of the library, $|N|$ is the maximum path length and $|FI|$ are the number of inputs to the tree. In traditional approaches, matches for every load value have to be determined and stored, and the library used for matching includes multiple sizes of each gate. In contrast, our approach stores solutions for all values of $|N|$ (which is small, on average), and the library has only one instance of each gate type. Since we store solutions for each path length, and each input to a fanout-free region, the storage requirement is $O(|V|\cdot N\cdot|FI|)$. Note that this is a very loose upper bound. We show in the results section that N is relatively small, and while $|FI|$ can be exponentially large in theory ($O(2^P)$, if the entire fanout-free region is a tree of 2-input gates for a path length of P from input to output), in practice, it is much smaller.

Calculating the Delay-C_{in} curves dominates the running time of our algorithm. The time complexity of this step is $O(|FI|\cdot|c_{in}|^2\cdot|FO|^2)$, where $|c_{in}|$ is the number of possible values for input capacitances, and $|FO|$ is the number of fanouts at a multiple fanout point. This bound too, is very loose, and for benchmark circuits, the running time is of the same order as that of SIS.

V. Results

In order to validate our approach, we have implemented Algorithm 4 and used it to map ISCAS and MCNC combinational benchmark circuits. These results were compared with SIS [17]. The library used for SIS was generated by calibrating INV, 2-, 3- and 4-input NAND and NOR gates, and a variety of AOI and OAI gates on a 0.1μ technology using the Berkeley Predictive Technology Model[3] [18]. Twenty sizes of each gate were generated, for a total library size of approximately 400 elements. These gates were also calibrated in order to obtain the logical effort and parasitic delays, which constitute the library used by our algorithm, with 23 elements, one for each gate type. In our approach, calculating gate sizes as described in Section IV can lead arbitrary values (less than the largest gate size). In order to make a fair comparison with SIS, gate sizes are normalized to the 20 sizes of each gate that are used by SIS.

Table I presents structural statistics of the benchmark circuits used in our experiments. For each circuit, we list the size, as determined by the number of gates, and the number of trees, after the circuit has been broken into fanout-free regions. Next, we present the minimum, maximum and average values of the sizes of the trees, the number of fanins and fanouts of each tree, and the path lengths within each tree. The traditional technology mapping approach maps each tree separately, whereas our approach deals with the entire circuit as a whole. Consequently, the running time of our algorithm depends not only on the sizes of each tree, but also on the number of fanins, fanouts and the path lengths within each tree. As can be seen from Table I, in the worst case, each of these can be large, however the average case, listed in the last row is much more tractable. For example, the circuit pair has a tree with 41 outputs. Combining the Delay-C_{in} curves of these outputs is expensive if these curves have approximately similar delay values. If this were the case for all trees, the runtime would be prohibitive. However, other trees in this circuit have much smaller fanouts, and an average

[3]Available from http://www-device.eecs.berkeley.edu/~ptm.

TABLE I
CIRCUIT STATISTICS

Circuit	Number of		Tree Size Statistics			|FI| Statistics			|FO| Statistics			Path Length Statistics		
	Gates	Trees	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
C17	35	3	1	7	4.66	1	2	1.66	2	2	2.00	1	5	2.20
C432	1071	75	1	300	12.06	1	63	3.37	2	28	3.44	1	23	2.95
C499	2037	155	1	82	10.25	1	16	3.05	2	12	3.61	1	15	3.03
C880	1841	115	1	71	8.87	1	14	2.69	2	9	3.71	1	22	3.46
C1355	2517	259	1	82	7.98	1	16	2.62	2	12	2.96	1	15	3.10
C1908	3145	188	1	124	13.13	1	26	3.57	2	25	4.23	1	20	3.68
C2670	4497	216	1	354	19.89	1	72	4.90	2	56	4.62	1	27	3.60
C3540	6428	320	1	318	19.09	1	62	4.75	2	41	4.86	1	22	3.88
C5315	10526	441	1	93	16.11	1	19	4.23	2	70	5.58	1	19	3.56
C6288	10029	1425	1	13	6.76	1	4	2.58	2	16	2.65	1	7	2.81
C7552	12869	704	1	164	16.47	1	35	4.37	2	230	4.59	1	23	3.80
9symml	1143	17	1	24	7.70	1	6	2.52	2	32	14.29	1	34	3.36
alu2	1919	68	1	306	18.73	1	63	4.76	2	39	6.63	1	33	3.32
apex6	3542	205	1	127	4.62	1	26	1.80	2	34	4.07	1	21	2.85
b9	674	28	1	85	10.14	1	17	2.82	2	18	4.60	1	23	3.20
cc	380	13	1	11	2.23	1	3	1.23	2	14	6.00	1	13	2.74
cm138a	148	4	1	13	4.00	1	3	1.50	8	8	8.00	1	8	4.00
count	637	34	1	5	2.73	1	2	1.44	2	32	4.17	1	11	3.23
cmb	267	13	1	118	10.00	1	25	2.84	2	6	3.76	1	26	3.47
decod	296	7	1	6	2.28	1	2	1.28	2	16	9.71	1	9	3.96
example2	1519	62	1	35	5.95	1	7	2.06	2	39	5.14	1	15	3.49
f51m	629	22	1	21	6.40	1	5	2.18	2	23	7.09	1	20	3.74
frg1	568	26	1	19	2.11	1	5	1.26	2	9	4.53	1	13	1.69
i5	2020	127	1	80	4.88	1	15	1.74	2	10	3.28	1	15	3.25
pair	8098	366	1	47	9.05	1	10	2.72	2	41	5.26	1	34	4.15
pcler8	400	25	1	12	5.00	1	3	1.72	2	9	3.44	1	12	3.00
t	35	3	1	7	3.00	1	2	1.33	2	2	2.00	1	8	3.00
ttt2	1375	28	1	20	4.39	1	5	1.71	2	25	10.57	1	25	4.80
vda	6716	107	1	57	27.00	1	11	5.95	2	92	13.35	1	23	6.34
x1	2073	47	1	4	1.19	1	2	1.06	2	34	9.27	0	24	2.89
z4ml	283	17	1	20	6.35	1	5	2.23	2	10	4.35	1	11	2.60
Average			Tree Size 9.334			|FI| 2.748			|FO| 5.647			Path Length 3.392		

fanout of 5.26 is tractablt. Similarly, the values of the other structural parameters presented in Table I affect the runtime of our algorithm. MELT determines and store matches for all values of path lengths for each input of fanout-free regions of the circuit. These are then examined to obtain the minimum achievable delays for the fanout-free regions. Therefore, having long path lengths and a large number of inputs can lead to large run times. As before, though some paths can be large, and a few trees have a large number of fanins, the averages are skewed towards small values.

The results of mapping these circuits are as shown in Table II. The first column lists the benchmark circuit. The next two, under the title SIS show the best delay obtained for each circuit using the command `map -n 1` in SIS, the area of the final solution and the corresponding running time, T, in seconds. The performance of MELT for the same circuits is as shown. On average, our algorithm generates circuits that are 37.20% faster than those obtained using SIS. Interestingly, MELT also has an average area improvement of 32.99%. During the covering step, the load at multiple fanout points is accurately known, as is the optimal electrical effort for individual segments, which is not taken into account by SIS. This leads to a higher incidence of complex gates in the MELT solution. The area tradeoff between using complex and simple gates depends on the sizes of each gate being selected. Smaller sizes of complex gates such as OAIs are more area-efficient than the equivalent circuit using simple gates such as NANDs, NORs and INVs. However, the larger sizes of these complex gates occupy more area than the equivalent circuit using simple gates. Thus, while we usually have an area improvement for most circuits, for some circuits, such as count, i5 and t, we obtain more expensive (albeit faster) solutions from MELT. In the case of C17 and C6288, the circuits selected are very similar, and consist largely of NAND2 gates. In this case, the gate sizes selected in MELT are larger than those selected by SIS. Once again, it is the electrical effort that guides this size selection. While we obtain mapped circuits that are faster, the area overhead in these cases is significant.

It is important to point out that it is quite understandable that the area numbers for our algorithm to be higher, simply because MELT only optimizes the delay, and does not explicitly optimize the area. Such a delay minimizer is very useful in practice, since it allows a designer to determine the best possible performance that can be achieved by a circuit. It is easily seen that in terms of delay, MELT always outperforms SIS, and the improvement over SIS ranges from just over 1% to nearly 80%.

This wide range in the achievable improvement can be explained by the circuit characteristics described in Table I. For example, C6288 has a large number of fanout-free regions with small average path lengths, as compared to other circuits. For small path lengths, the effect of varying the electrical effort is limited, which in turn restricts the freedom that our algorithm has, and results in solutions that are very similar to those of that would be obtained by traditional methods, and therefore only a 1.31% improvement in delay is seen in this circuit. In contrast, C7552 has fewer, but larger fanout-free regions, which results in better mapped solutions, leading to a solution from MELT that has a delay 39.88% better than that obtained from SIS.

TABLE II
TECHNOLOGY MAPPING: SIS VS. MELT

Circuit	SIS			MELT			% Change in	
	Delay(ps)	Area	T(s)	Delay(ps)	Area	T(s)	Delay	Area
C17	79.91	18.96	0.10	65.72	47.61	0.04	17.76	-151.11
C432	1629.21	1609.07	6.84	795.93	1407.17	4.03	51.15	12.55
C499	822.77	2143.44	11.53	658.15	2813.28	3.20	20.01	-31.25
C880	700.98	1634.78	7.58	643.43	1399.71	2.20	8.21	14.38
C1355	854.79	2527.04	11.44	678.19	2581.55	4.55	20.66	-2.16
C1908	1289.69	3431.64	21.99	868.42	2402.54	4.23	32.66	29.99
C2670	2161.43	5664.06	41.95	868.82	3515.99	8.19	59.80	37.92
C3540	2624.79	9720.64	45.40	1218.21	4650.69	12.90	53.59	52.16
C5315	1709.62	13790.11	103.58	971.36	7016.89	13.49	43.18	49.12
C6288	2931.69	11358.46	50.51	2893.31	18008.40	17.94	1.31	-58.55
C7552	1825.71	18687.65	183.32	1097.69	6903.13	19.64	39.88	63.06
9symml	1229.74	2301.95	9.76	346.78	610.05	0.55	71.80	73.50
alu2	2357.10	4123.53	21.19	1137.77	1449.93	2.76	51.73	64.84
apex6	769.49	3691.38	16.93	388.89	3687.13	3.86	49.46	0.12
b9	440.00	604.68	3.70	227.89	684.65	0.56	48.21	-13.23
cc	307.25	272.07	2.23	157.22	232.89	0.25	48.83	14.40
cm138a	195.23	125.88	0.71	120.11	125.64	0.16	38.48	0.19
cmb	228.95	241.70	1.03	216.70	258.88	0.29	5.35	-7.11
count	907.22	553.14	2.84	592.17	854.11	0.38	34.73	-54.41
decod	326.95	232.98	2.50	98.50	291.79	0.26	69.87	-25.24
example2	711.10	1404.83	8.51	331.66	1179.15	1.46	53.36	16.06
f51m	753.01	897.81	7.20	344.53	366.36	0.46	54.25	59.19
frg1	472.26	950.53	2.54	379.31	494.52	0.50	19.68	47.97
i5	392.54	1419.17	6.81	222.76	2330.55	1.93	43.25	-64.22
pair	1224.90	8675.52	53.86	814.31	7053.61	11.78	33.52	18.70
pcler8	615.45	405.61	1.51	331.04	551.60	0.33	46.21	-35.99
t	77.10	19.44	0.11	74.92	56.22	0.03	2.82	-189.20
ttt2	824.10	1724.02	15.50	279.97	854.25	0.68	66.03	50.45
vda	2193.22	11328.42	70.82	443.30	1888.76	20.75	79.79	83.33
x1	1055.87	2632.88	16.95	343.81	1444.36	1.21	67.44	45.14
z4ml	301.13	256.44	1.58	198.91	189.20	0.31	33.95	26.22
Average							37.20	32.99

VI. CONCLUSION AND FUTURE DIRECTIONS

This paper presents a new approach to technology mapping, based on the theory of logical effort. Most of the improvement obtained by our algorithm is due to the solution of the load-distribution problem, which allows for accurate assignment of capacitances at multiple fanout points. This leads to better selection of matches, since the exact load to be driven is known. We observe an average improvement of 37.20% in terms of delay and 32.99% in terms of area, as compared to SIS.

In [19], [20], all possible decompositions of circuits are considered during the matching step. The algorithm divides the circuit into disjoint *ugates*, and applies technology mapping to each such *ugate*. Our algorithm can be extended to generate matches in each *ugate*, and calculate Delay-C_{in} curves by traversing the ugates. This approach can also be applied to DAG-mapping [2], which allows matches across tree boundaries, and therefore can generate better solutions. Here, multiple fanout points are not well defined initially. However, once the matching has been done, the fanout points are specified and the Delay-C_{in} curves can be calculated as before.

APPENDIX
PROOF OF LEMMA 1

We first prove the case of symmetric gates, in which the delay characteristics of each input pin to the output of the gate are the same. The proof for the case of asymmetric gates is similar, and follows from the proof for symmetric gates.

Consider the situation where we have a match at some gate t, with r inputs, $I_1, I_2, \ldots, I_r$, each having cumulative logical effort for path of length n from the primary inputs $G_{I_1}[n], G_{I_2}[n], \ldots G_{I_r}[n]$. Since gate t is symmetric, the load being driven by each of $I_1, I_2, \ldots, I_r$ is equal, and is c_{in_t}, a value that is yet to be determined. Let I_c be the critical input, and let I_j denote the other non-critical inputs. As mentioned previously, I_c being the critical input implies that $G_{I_c}[n] \geq G_{I_j}[n]\ \forall\ j$. In this case, we select $G_{I_c}[n]$ to be multiplied with the gate effort of the match at t, g_{m_t} in order to obtain $G_t[n+1]$. This means that when the segment is sized, the size of the match at gate t (which determines the load c_{in_t} at the output of any I_j) will be determined by the value of $G_{I_c}[n]$, and this size will be different from the size determined if we had selected $G_{I_j}[n]$. We show that this is in fact the correct choice to make.

As mentioned previously, the sizes of gates are determined by applying Equation (5) in a backward traversal. If the load at the primary output is C_L, and there are k gates from gate t to the primary output in the mapped solution, Equation (5) can be applied to each gate successively, to obtain

$$c_{in_t} = \frac{\prod_k g_k}{\hat{f}^k} \cdot C_L \tag{9}$$

Note that the stage effort for optimal delay is in the denominator of Equation (9), and $\hat{f} = F^{\frac{1}{N}} = (G \cdot H)^{\frac{1}{N}}$. Therefore, choosing $G_{I_c}[n]$ induces a size on gate t that is smaller than that we would have obtained by using $G_{I_j}[n]$. This means that the delay $G_{I_j}[n]$ would have induced (say D_{I_j}) would depend on a *larger* size of gate t. Since t is now smaller than anticipated by gate I_j, its load on I_j is smaller, and hence the delay of the branch from an input to gate I_j does not increase (from the value it would have been, if the solution corresponding to I_j had been used to size t) by taking the choice of $G_{I_c}[n]$, i.e., I_c is still the critical input of t.

We now turn to the general case of asymmetric gates. In this case, the logical effort of each input of the gate to the output depends on the functionality. However, we show that the assertion of Lemma 1 still holds.

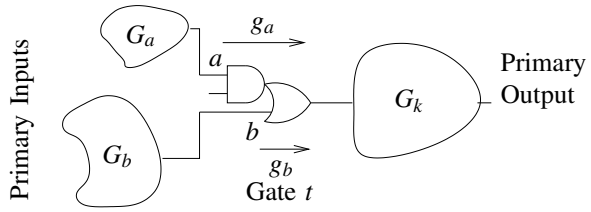


Fig. 8. Proof of Lemma 1 for Asymmetrical Gates

Consider asymmetric gate t with two of its inputs, a and b having logical efforts g_a and g_b respectively. Let the cumulative logical effort up to each input be G_a and G_b, and the cumulative logical effort from the output of t to a primary output be G_k. Since gate t is asymmetric, the capacitance at each input is different, but this also implies that the logical efforts are in the same ratio i.e., if $c_{in_b} = z \times c_{in_a}$, then $g_b = z \times g_a$ (this follows from the definition of logical effort). There are two cases to consider, as follows.

1) $G_a > G_b$: In this case, the solution at input a is selected as it is considered to be critical. Sizing gate t according to this solution will imply some capacitance $c^*_{in_b}$ at input b, and we need to show that $c^*_{in_b} \leq c_{in_b}$.

$$\begin{aligned} c_{in_a} &= \frac{G_k \cdot g_a}{\hat{f}_a^k} \cdot C_L \\ c^*_{in_b} &= z \times c_{in_a} \\ &= z \times \frac{G_k \cdot g_a}{\hat{f}_a^k} \cdot C_L \\ &= \frac{G_k \cdot g_b}{\hat{f}_a^k} \cdot C_L \\ c_{in_b} &= \frac{G_k \cdot g_b}{\hat{f}_b^k} \cdot C_L \end{aligned}$$

Since $G_a > G_b$, $\hat{f}_a > \hat{f}_b$ and $c^*_{in_b} < c_{in_b}$.

2) $G_b > G_a$: As in the previous case, the solution at input b is selected, which implies some capacitance $c^*_{in_a}$ at input a. We need to show that $c^*_{in_a} \leq c_{in_a}$.

$$\begin{aligned} c_{in_b} &= \frac{G_k \cdot g_b}{\hat{f}_b^k} \cdot C_L \\ c^*_{in_a} &= \frac{c_{in_b}}{z} \\ &= \frac{G_k \cdot g_b}{\hat{f}_b^k \times z} \cdot C_L \\ &= \frac{G_k \cdot g_a}{\hat{f}_b^k} \cdot C_L \\ c_{in_a} &= \frac{G_k \cdot g_a}{\hat{f}_a^k} \cdot C_L \end{aligned}$$

Since $G_b > G_a$, $\hat{f}_b > \hat{f}_a$ and $c^*_{in_a} < c_{in_a}$.

Thus, in both cases, the non-critical input eventually drives a smaller load than anticipated, and therefore the delay at the non-critical input does not increase to a value greater than that of the critical input.

REFERENCES

[1] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," in *Proceedings of the IEEE/ACM Design Automation Conference*, June 1987, pp. 341–347.

[2] Y. Kukimoto, R. K. Brayton, and P. Sawkar, "Delay-Optimal Technology Mapping by DAG Covering," in *Proceedings of the IEEE/ACM Design Automation Conference*, June 1998, pp. 348–351.

[3] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, "Performance-Oriented Technology Mapping," in *Proceedings of the 6th MIT Conf. on Advanced Research in VLSI*, 1990, pp. 79–97.

[4] J. Grodstein, E. Lehman, H. Harkness, B. Grundmann, and Y. Watanabe, "A Delay Model for Logic Synthesis of Continuously-Sized Networks," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, Nov. 1995, pp. 458–462.

[5] L. Stok, M. A. Iyer, and A. J. Sullivan, "Wavefront Technology Mapping," in *Proceedings of the Design, Automation and Test in Europe Conference*, Mar. 1999, pp. 531–536.

[6] B. Hu, Y. Watanabe, A. Kondratyev, and M. Marek-Sadowska, "Gain-Based Technology Mapping for Discrete-Size Cell Libraries," in *Proceedings of the IEEE/ACM Design Automation Conference*, June 2003, pp. 574–579.

[7] R. F. Sproull and I. E. Sutherland, "Theory of Logical Effort: Designing for Speed on the Back of an Envelope," in *IEEE Advanced Research in VLSI*, 1991, pp. 1–16.

[8] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*. San Fransisco, CA: Morgan Kaufmann, 1999.

[9] F. Beeftink, P. Kudva, D. Kung, and L. Stok, "Gate-Size Selection for Standard Cell Libraries," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, Nov. 1998, pp. 545–550.

[10] W. Donath, P. Kudva, L. Stok, P. Villarrubia, L. Reddy, A. Sullivan, and K. Chakraborty, "Transformational Placement and Synthesis," in *Proceedings of the Design, Automation and Test in Europe Conference*, Mar. 2000, pp. 194–201.

[11] K. Sulimma, I. Neumann, L. van Ginneken, and W. Kunz, "Improving Placement under the Constant Delay Model," in *Proceedings of the Design, Automation and Test in Europe Conference*, Mar. 2002, pp. 677–682.

[12] L. Stok, D. S. Kung, D. Brand, A. D. Drumm, A. J. Sullivan, L. N. Reddy, N. Hieter, D. J. Geiger, H. H. Chao, and P. J. Osler, "BooleDozer: Logic Synthesis for ASICs," *IBM Journal of Research and Development*, vol. 40, no. 4, pp. 407–430, 1996.

[13] "Gain Based Synthesis: Speeding RTL to Silicon," 2002, magma Design Automation white paper. Available at http://www.magma-da.com/c/@57hzNi1ExOwpA/Pages/Gainbasedoverview.html.

[14] S. K. Karandikar and S. S. Sapatnekar, "Fast Comparisons of Circuit Implementations," in *Proceedings of the Design, Automation and Test in Europe Conference*, Feb. 2004, pp. 910–915.

[15] ——, "Fast Estimation of Area-Delay Tradeoffs in Circuit Sizing," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, June 2005, pp. 3575–2578.

[16] ——, "Fast Comparisons of Circuit Implementations," *IEEE Transactions on VLSI Systems*, vol. 13, no. 12, pp. 1329–1339, Dec. 2006.

[17] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Electronics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Tech. Rep. UCB/ERL M92/41, May 1992.

[18] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Design," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2000, pp. 201–204.

[19] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic Decomposition during Technology Mapping," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, Nov. 1995, pp. 264–271.

[20] ——, "Logic Decomposition During Technology Mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 8, pp. 813–834, Aug. 1997.