Computing the Area versus Delay Trade-off Curves in Technology Mapping [1]

Kamal Chaudhary
Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124

Massoud Pedram
University of Southern California
Department of EE - Systems
Los Angeles, CA 90089

[1]This research was supported in part by NSF under contract No. MIP-9211668 and by SRC under contract No. 94-DJ-559.

List of Figures

List of Tables

Abstract

We examine the problem of mapping a Boolean network using gates from a finite size cell library. The objective is to minimize the total gate area subject to constraints on signal arrival time at the primary outputs. Our approach consists of two steps: In the first step, we compute delay functions (which capture gate area – arrival time tradeoffs) at all nodes in the network, and in the second step we generate the mapping solution based on the computed delay functions and the required times at the primary outputs. For a NAND-decomposed tree, subject to load calculation errors, this two step approach finds the minimum area mapping satisfying a delay constraint if such solution exists. The algorithm has polynomial run time on a node-balanced tree and is easily extended to mapping a directed acyclic graph (DAG). We also show how to account for the wire delays during the delay function computation step. Our results compare favorably with those of MIS2.2 mapper.

1 Introduction

The goal of logic synthesis is to produce a circuit which satisfies a set of logic equations, occupies minimal area and meets the timing constraints. Most logic synthesis systems currently available split this task into two phases – a technology independent phase and a technology dependent phase. In the first phase, transformations are applied on a Boolean network to find a representation with the least number of literals in the factored form. Additional timing optimization transformations are applied on this minimal area network to improve circuit performance. The role of the technology-dependent phase is to finish the synthesis of the circuit by performing the final gate selection from a target library. The technology-dependent phase is, to a large extent, constrained by the structure of the optimized Boolean network.

1.1 Prior Work

The technology mapping problem can be stated as follows: Given a Boolean network representing a combinational logic circuit optimized by technology independent synthesis procedures and a target library, we bind nodes in the network to gates in the library such that area of the final implementation is minimized and timing constraints are satisfied.

A successful and efficient solution to the minimum area mapping problem was suggested by K. Keutzer and implemented in programs such as DAGON [10] and MIS [7]. The idea is to reduce technology mapping to DAG covering and to approximate DAG covering by a sequence of tree coverings which can be performed optimally using dynamic programming as follows. A set of base functions is chosen, such as a 2-input nand gate and an inverter. The optimized logic equations (obtained from technology independent optimization) are converted into a graph where each node is one of the base functions. This graph is called the *subject graph*. Each library gate is also represented by a graph consisting of only base functions. Each such graph is called a *pattern graph*. (Each library gate may have many different pattern graphs.) The technology mapping problem is then defined as the problem of finding a minimum cost covering of the subject graph by choosing from the collection of pattern graphs for all gates in the library. For area optimization, the cost of a cover is defined as the sum of gate areas. For minimum delay optimization [18], the cost of a cover is defined as the critical path delay of the resulting circuit.

This approach is extended in [24] to solve the technology mapping problem minimizing area under delay constraints as follows. The authors first compute a range of "interesting" values for the required times at each node (by finding the minimum area and the minimum delay mapping solutions) and then divide this range into equal intervals. The best mapping solution for each of the required times are generated and stored at the node during a postorder traversal (from primary inputs to primary outputs) of the tree. The final mapping solution is generated during a preorder traversal (from primary outputs toward primary inputs) of the tree. In order to obtain high-quality mapping solutions, this method requires a small time step resulting in large number of delay-area points. In contrast, our method [4] works with the arrival times (as opposed to the required times), keeps all (and only) non-inferior delay-area points, and does not need an a priori range of interest for arrival times.

Technology decomposition (the procedure for converting an optimized Boolean network into a NAND-decomposed network) is the precursor to the technology mapping step. It is an open problem to determine which of the possible NAND-decomposed networks yields an optimum solution when an optimum covering algorithm is applied [3]. Different decomposition schemes for minimizing area [18], minimizing delay [14, 26], or reducing routing complexity [16] have been introduced by various authors. In this paper, we assume that the DAG has already been decomposed into two-input NAND and inverter gates.

Other technology mapping programs based on rules [6, 2], heuristic gate merging and sizing [11], algebraic identity [14], and Boolean matching [12] have also been proposed in the literature.

1.2 Overview and Organization of the Paper

In this paper, we present an efficient algorithm for generating a technology mapping solution with minimum gate area subject to given delay constraints. Our approach consists of two steps: In the first step, we compute delay functions (which capture arrival time – gate area tradeoffs) at all nodes of a NAND-decomposed network, and in the second step we generate the mapping solution based on the computed delay functions and the required times at the primary outputs.

The paper is organized as follows. In Section 2, we introduce some terminology and describe the timing model. Section 3 presents details of our algorithm. Sections 4 and 5 are devoted to the extension from trees to general DAGs and the complexity analysis. Section 6 describes an extension to account for the wire delays during the delay function computation step. We present our results and concluding remarks in Sections 7 and 8.

2 Terminology and Timing Model

Consider a match g at node n of a NAND-decomposed tree. The inputs to node n consist of nodes n_i that fanout to node n (that is, $n = n'_1 + n'_2$ if n has two inputs or $n = n'_1$ if n has a single input).

The nodes which are covered by match g are denoted by $merged(n, g)$. The nodes which are not in $merged(n, g)$ but fanin to $merged(n, g)$ are denoted by $inputs(n, g)$. The *mapped-parent*(n_i) is the set of nodes n_j for which there exists a matching gate g such that $n_i \in inputs(n_j, g)$. Note that given node n and gate g matching at n, $inputs(n, g)$ are uniquely determined. However, n may have many distinct mapped parents (Figure 1).

Figure 1 goes here.

With each node in the network we store a *delay curve* (also referred as a *delay function*). A point on the curve represents the arrival time at the output of the node and the total gate area which is required to map its transitive fanin cone up to (and including) the node. In addition to the area and delay value, the matching gate and input bindings for the match are also stored with each point on the curve. Points on the curve represent various mapping solutions with different tradeoffs between area and speed. We are interested in a mapping with minimum area satisfying delay requirements. Consequently, we can drop point P_1 on the curve if there exists another point P_2 on the curve with lower area but equal or lower delay. This is possible because the solution associated with P_2 is superior to the solution associated with P_1 in terms of area, delay or both. By dropping points, the delay curve can always be made monotonically non-increasing without loss of optimality. We would refer to P_1 as an *inferior point*. Point $P^\star = (t^\star, a^\star)$ is a *non-inferior point* if and only if there does not exist a point $P = (t, a)$ such that either $t \leq t^\star, a < a^\star$ or $t < t^\star, a \leq a^\star$.

Lemma 2.1 *The delay function for a node contains the set of all non-inferior points and is monotonically non-increasing.*

In addition, if the difference in delay among two points is small (according to some user-specified parameter ϵ), we drop the point with higher area without any noticeable impact on the quality of the result. Similarly, points which are close in terms of their areas are merged together.

For the delay computation, we have adopted the pin-dependent MIS library delay model as described next. Suppose that gate g has matched at node n, then the output arrival time at n is given by:

$$arrival(n, g, C_n) = max_{n_i \in inputs(n,g)}(\tau_{i,g} + R_{i,g} \times C_n + arrival(n_i, g_i, C_{n_i}))$$

where $\tau_{i,g}$ is the *intrinsic* gate delay from input n_i to output of g, $R_{i,g}$ is the *drive* resistance of g corresponding to a signal transition at input n_i, C_n is the load capacitance seen at n, $arrival(n_i, g_i, C_{n_i})$

is the arrival time at input n_i corresponding to load C_{n_i} seen at that input, and g_i is the best match found at input n_i.

The above equation can be easily modified to calculate the rise or fall delays based on the phase of the gate (INVERTING, NON-INVERTING or UNKNOWN), the corresponding rise and fall delay parameters for the inputs, and the output load. To simplify the exposition however, we will use the above generic delay equation.

This delay equation is based on a static timing analysis model which ignores the false path problem. Doing true timing analysis ([13, 5]) during technology mapping will significantly increase the computational and space complexities of our proposed algorithm as follows: 1) Timing analysis will have to be performed in a (either explicit or implicit) path-based manner rather than the block-oriented manner in which it is currently performed; 2) The new delay equation will not satisfy the principle of dynamic programming and hence the number of area-delay tradeoff points that will have to be stored in the delay functions of nodes will become exponential (as the points will have to be annotated with the sub-path that generates them and the sensitization conditions along that sub-path). In this paper, we will not consider this issue further.

3 Tree Mapping

In this section, we focus on tree mapping. Later, we shall describe extensions to DAG mapping. In particular, we describe two tree-traversal operations which are applied to a NAND-decomposed tree in order to obtain a technology mapping solution which minimizes the total gate area while satisfying the timing constraints.

First, a postorder traversal is used to determine a set of possible arrival times at the root of the tree. Once the user specifies a single required time, a second, preorder traversal is performed to determine a specific technology mapping solution. This scheme is similar to that proposed in [23] in order to solve the optimal orientation problem for a slicing tree of macro-cells.

We begin by stating that the possible accumulated gate areas at each node can be described as a function of the arrival times at the node. The accumulated gate area is the total area used by the gates which have matched nodes in the transitive fanin cone of the node. The arrival time is the earliest time at which the signal at the output of the node settles within 50% of its final value (due to a signal transition at some primary input). The delay function is therefore represented by a set of ordered pairs of real positive numbers (t, a), where a piecewise linear function $a = f(t)$ can be constructed which describes the graph of all possible accumulated gate areas. This function describes all possible arrival time-area tradeoffs at a given node. The delay function at an input node of the NAND-decomposed tree consists of ordered pairs (t, a) where t and a have been specified by the user (in case of primary inputs of the network) or have been previously computed (in case that inputs to this tree are outputs of other trees).

3.1 Postorder Traversal

On the first traversal, we begin at the leaf nodes of the NAND-decomposed tree. Since each leaf node possesses a set of possible arrival time-area points which are reflected in its delay function, the delay function at any *mapped-parent*(n) must also reflect these possible arrival time-area tradeoffs. A postorder traversal of the NAND-decomposed tree is performed, where for each node n and for each gate g matching at n, a new delay function is produced by appropriately merging the delay functions at the $inputs(n, g)$. Merging must occur in the common region among all delay functions in order to ensure that the resulting merged function reflects feasible matches at the $inputs(n, g)$. The delay functions for successive gates g matching at n are then merged by applying a *lower-bound merge* operation on the corresponding delay functions. At a given node n, the resulting delay function will describe the arrival time-area tradeoffs in propagating a signal from the tree inputs to the output of n.

Figure 2 goes here.

To illustrate the delay function computation procedure, consider the example in Figure 2. It shows the computation of the delay function for match g at node D. The inputs to the match are nodes A and B. The delay functions for A and B are known at this time. To compute a point on the delay function for node D, we select a point from delay function of inputs, say point a on delay curve of node A. The delay of point a is 3 units. So, we look for a point on the delay function of node B with delay less than 3 which has the minimum area. In this example, d is the desired point. We therefore combine points a and d to generate point a' on delay-curve(D), with

$$arrival(a') = arrival(a) + delay(g)$$

$$area(a') = area(a) + area(d) + area(gate).$$

Similarly, we generate all other points on the curve. Note, that there is no point on delay-curve(D) corresponding to the point e on delay-curve(B), as there is no point on delay-curve(A) which has delay less than or equal to delay(e).

Figure 3 goes here.

To illustrate the lower bound merging procedure, consider the example in Figure 3. Here, we have already generated the delay-curves for the matching gates g_1 and g_2 at node C. In order to obtain the composite delay curve at n, we must merge the two delay curves into one. This operation is simple since we only need to keep the non-inferior points on either curve. The minimum of the two delay-curves is computed, and information is attached to each point on the resulting delay curve indicating which gate alternative yields that point.

The delay function computation and merging are performed recursively until the root of the tree is reached. The resulting function is saved in the tree at its corresponding node. Thus, each node of the tree will have an associated delay function. The set of (t, a) pairs corresponding to the composite delay function at the root node will define a set of arrival time-area tradeoffs for the user to choose from.

3.2 Preorder Traversal

The user is allowed to select the arrival time-area tradeoff which is most suitable for his application. Given the required time t at the root of the tree, a suitable (t, a) point on the delay function for the root node is chosen. The gate g matching at the root that corresponds to this point and $inputs(root, g)$ are, thus, identified. The required times t_i at $inputs(root, g)$ are computed from t, g, and the observation that $inputs(root, g)$ must now drive gate g. The preorder traversal resumes at $inputs(root, g)$ where t_i is the constraining factor and a matching gate g_i with minimum a_i satisfying t_i is sought.

3.3 Timing Recalculation

The gate delay is a function of the load it is driving. During the postorder tree traversal, the output of current node n_i, is not mapped hence the load capacitance is unknown (unless, all the gates in the library have identical pin capacitances). This load cannot be taken to be zero as that will introduce excessive error during the postorder traversal; at the same time, the average number of pins per gate in a mapped circuit is between 2 and 3, therefore, we heuristically choose the load value to be that offered by the smallest two-input NAND gate in the library. When we come to a node $n \in$*mapped-parent*(n_i) with matching gate g, we know the exact load seen by n_i. This load is equal to the input capacitance of g and is, in general, different from the default load. Therefore, in order to calculate the arrival time at node n, the output arrival times for all nodes in $inputs(n, g)$ must be adjusted to account for the change in the load capacitance [15]. Similarly, during the preorder tree traversal, when a gate g is selected to match at n, the load seen by $inputs(n, g)$ must be recalculated.

In order to account for this load change (δ_i), the delay curves at the inputs have to be appropriately shifted. In particular, since the drive resistance of gate matching at n_i and giving rise to a point p_j on delay-curve of n_i is stored with that point, the delay shift is computed as $\delta_i \times p_j.gate.drive$. (See pseudo-code for compute_delay_curve and assign_best_gate algorithms. Details of timing recalculation are given for compute_delay_curve.)

```
function compute_delay_curve(n)
begin
  for each candidate match $g$ at the node $n$ do
    for each input $i$ of match $g$ do
      $\gamma_i = g.intrinsic_i + g.drive_i \times n.load$
      $\delta_i$ = calculate change in load
    end
    for each input $i$ of match $g$ do
      for each point $p_j$ on the delay curve of input $i$ do
        $np.delay = p_j.delay + \delta_i \times p_j.gate.drive + \gamma_i$
        feasible-flag = true
        for each input $k$ of match $g$ do
          if $k = i$, continue
          find $p_k^\star$ where $p_k^\star.area$ is minimum and
                $p_k^\star.delay \leq np.delay - \gamma_k$
          if no such $p_k^\star$ found
            feasible-flag = false
            break
          end
        end
        if feasible-flag = true
          $np.gate = g$
          $np.binding = inputs(n, g)$
          $np.area = g.area + \sum_k p_k^\star.area$
          insert $np$ in the delay-curve$(n)$
        end
      end
    end
  end
  sort delay curve of $n$ based on the delay
  delete the inferior points on the curve
  reduce the non-inferior points by merging
end
```

```
function assign_best_gate(n, t)
begin
    calculate the current load based on the partial mapping
    shift delay-curve($n$) to reflect the change in load
    find a point $p^\star$ on delay-curve($n$) which satisfies the
        required time $t$ and has minimum area
    $n.best\_gate = p^\star.gate$
    $n.best\_binding = p^\star.binding$
    for each input $i \in inputs(p^\star.gate, n)$ do
        $\gamma_i = g.intrinsic_i + g.drive_i \times n.load$
        assign_best_gate($n_i, t - \gamma_i$)
    end
end
```

3.4 Accounting for the Unknown Load Values

The shift in delay for a point is a function of change in the load and the matching gate's driving resistance at the point. Different points on a curve may shift by different amounts depending on the matching gate. Differential shift may make the curve non-monotonic. In the worst case, a previously inferior point on the curve might have become non-inferior, had it not been dropped earlier. This may cause an optimal mapping being rejected. A possible solution is not to drop inferior points from the delay curves till we reach the output node. This will require a large number of points being stored for each curve without much gain.

Theorem 3.1 *Let R_{max} and R_{min} be the maximum and minimum driving resistances, C_{max} and C_{min} the maximum and minimum loads among gates in the library, and ϵ the error tolerance. If $(R_{max} - R_{min}) \times (C_{max} - C_{min}) \leq \epsilon$, then no optimal solution is dropped.*

Proof Maximum delay shift among the points on the curve is given by $(R_{max} - R_{min}) \times (C_{max} - C_{min})$. If this quantity is $\leq \epsilon$, then the error will be within the specified tolerance. ■

Corollary 3.2 *If all the gates have the same pin capacitances, then the tree traversals will produce the optimum solution.*

Corollary 3.3 *If all the gates have the same drive resistances, then the tree traversals will produce the optimum solution.*

The other possible solution is to use a load bin method similar to that of MIS2.2. For each load bin, we store a delay curve. If the load bins are separated by less than $\epsilon/(R_{max} - R_{min})$, then the timing error will be less than ϵ. In practice, most of the libraries have a small number of gate series (e.g., performance- versus area-optimized, low-power versus high-power series). Within each series, the gates tend to have almost the same pin capacitances. Therefore, use of one load bin per gate series should be sufficient.

Note that during delay estimation we ignore the wire load (or alternatively, approximate it based on the expected average interconnect length and the capacitance per unit length of interconnect). In fact, wire load can vary by a large amount (compared to the variance in pin capacitance) depending on the placement and routing. Therefore, it does not pay much to improve the accuracy of computing gate loads while ignoring (or only roughly capturing) the wire loads.

4 DAG Mapping

Most of the real circuits are not trees, but general DAGs. The problem of mapping a DAG even for the constant load model is NP-hard [3]. Therefore, we resort to heuristics. One heuristic is to decompose the DAG into a number of trees such that the inputs for each tree come from other tree outputs or the primary inputs. During the delay curve computation step, entire trees are processed in postorder and delay curves are computed for each primary output of the DAG. During the gate assignment step, entire trees are mapped in preorder. This heuristic which does not allow mapping across tree boundaries is similar to that used by DAGON.

Alternatively, we could avoid decomposing the DAG into trees as follows. During the delay curve computation step, nodes are visited in postorder. For each node, we compute the delay curve as in case of trees. However, if the input for a candidate match at the node is coming from a multiple fanout node we divide the area contribution of that input by the fanout count of the input node. By reducing the area contribution we tend to favor a solution in which multiple fanout nodes are preserved after mapping, which reduces logic duplication and improve the final mapped area. This heuristic which permits tree boundary crossing was also implemented in the MIS mappers [7, 24]. This approach leads to smaller circuits, and is the one which we adopted for our mapper.

During the gate selection step, if we come to a node which has already been mapped, we check if the mapped solution at the node satisfies the timing requirement. If so, we keep the mapping;

otherwise, we replace it with another solution from the delay curve which satisfies the current timing requirement and has minimum area. The new solution may have higher area compared to the previous solution. Note that satisfying the current timing can only decrease the delay for the previous cones, although it may increase the total gate area.

The solution for circuits with multiple outputs also depends on the order in which the output cones are processed. During the delay curve generation step, when we are computing the signal arrival time for a match g at node n, we need to recalculate the load seen at $inputs(n, g)$. For $n_i \in inputs(n, g)$, some of the fanouts of node n_i (other than g) may have already been mapped (because they are part of a logic cone which has been processed), and hence, the contribution of these fanouts to the load can be calculated exactly. This incremental load recalculation will result in more accurate arrival time calculation at the output of n. Similar incremental load recalculation is applied during the gate assignment step.

```
**function** technology_map($\eta$, $\theta$)
*$\eta$ is a NAND-decomposed Boolean network*
*$\theta$ is a vector of required times at primary outputs*
**begin**
    **for** each node $n \in \eta$ (in postorder) **do**
        compute_delay_curve($n$)
    **end**
    **for** each primary output $po \in \eta$ **do**
        assign_best_gate($po, t_{po}$)
    **end**
**end**
```

5 Complexity Analysis

Consider a gate g (with k inputs) matching at node n where input i has N_i points on its delay curve. The delay curve corresponding to match g at node n has $N = \sum_{i=0}^{k} N_i$ points in the worst case. The time required to generate each point, assuming that delay curve for each input is sorted, is $O(klog(N_{max}))$ (time for binary search) where N_{max} is the maximum N_i. Thus, the total time for generating delay curve per candidate match is $O(kNlog(N_{max}))$.

For a finite size library, the maximum number of gates that can match at a node n is bounded which means that the number of points on the delay-curve(n) will remain linear in the total number of points on the delay-curve of $inputs(n, g)$ for various matching gates. Therefore, the number of points increases linearly from one level to another. Despite this, the number of points could still grow exponentially in terms of the number of levels in the tree. However, if the tree is node-balanced (its height is logarithmic in the number of its leaf nodes), then the number of points will remain polynomial. In practice, the increase in number of points is even lower due to the fact that a large number of points generated are inferior points which are dropped and not propagated to higher levels.

It is observed that the range of areas generated for various solutions varies only by a factor of two, which means that if we use only 50 points at each node, the solutions produced will be at most 2% poorer in the area compared to the case where unbounded number of points are allowed. With a fixed upper bound P on the number of points, the time to generate delay curve becomes $O(k^2 Plog(P))$ which is a constant since the number of inputs of any gate in the library is bounded.

6 Placement-Driven Mapping

6.1 Motivation

Interconnections are becoming a major concern in today's high-performance, high-density ASIC designs because the distributed RC time delay of these lines increases rapidly as chip sizes grow and minimum feature sizes shrink [1]. With recent studies [20, 9] indicating that interconnections occupy more than half the total chip area and account for a significant part of the chip delay, it is appropriate that wiring is integrated into the cost function for logic synthesis.

To elaborate on the importance of the wire load, consider a two-input NAND gate driving an inverter gate through 0.2 cm of aluminum interconnect (2 μm wide, 0.5 μm thick, with a 1.0 μm thick field oxide beneath it). 0.2 cm is the expected length of a *local* interconnect line on a $2cm \times 2cm$ chip [1]. We calculate the rise time (to 50% of its final value) at the input of the inverter gate using two methods. One method ignores the capacitance of the interconnect line and uses

$$delay = \tau_g + R_g \times C_g = 0.4ns$$

where τ_g is the intrinsic gate delay, R_g is the on-resistance of the driver gate, and C_g is the input capacitance of the fanout gate. The second method [19] uses

$$delay = \tau_g + R_g \times (C_g + C_{unit} \times length) = 1.0ns$$

where C_{unit} is the interconnect capacitance per unit length and $length$ is the interconnect length.[1] Gate and interconnect parameters are taken from data sheets for an industrial 1.0 micron ASIC library: $\tau_g = 0.3ns$, $R_g = 1K\Omega$, $C_g = 0.1pF$, $C_{unit} = 3pF/cm$. The delay calculations clearly show that the interconnect capacitance dominates the gate input capacitance.

In summary, with the existing technology, the capacitive term is dominated by the capacitance between the interconnection and substrate. For local aluminum lines, the resistive term is dominated by the on-resistance of the MOS transistor. As the chip dimension increases and the minimum feature size decreases, the interconnection capacitance bottoms at about 1 - 2 pF/cm while the input gate capacitance decreases. Therefore, the RC delay of interconnect lines will become even more dominant in the future.

In [15] an attempt is made to increase the interaction between logic synthesis and technology mapping. The idea is to generate a "companion" placement solution for the circuit before it is mapped. This placement is then used to evaluate the cost of a matching gate during the mapping process. The placement is dynamically updated in order to maintain the correspondence between the logic and layout representations. In the end, a mapped network and a corresponding placement solution are generated. The placement solution is then globally relaxed in order to produce a feasible placement according to the target layout style (e.g., standard-cell or sea-of-gates). Using these techniques, circuits with smaller area and higher performance have been synthesized [15].

6.2 Accounting for the Wire Delays

For submicron technologies, the effect of interconnect on circuit delay is of more importance than its effect on the circuit area. Therefore, we only consider the former effect here. The latter effect can be easily captured in a similar fashion. It is straight-forward to incorporate the wire cost into the area-delay mapper as described next.

The delay function at each node now consists of a set of non-inferior points $\check{P} = (\check{t}, a)$ where $\check{t}$ is a number representing the gate and wire delay, and a is the area. The load at the output of a node consists of two components: the gate capacitance of fanout nodes and the wire load. The latter is calculated as the product of the wire length and the capacitance per unit length of interconnect. Node positions are needed to compute the wire length. These position are however known after the initial placement of the unmapped network. Consequently, the wire length can be calculated using a number of different models. These models include the star connection model, the enclosing

[1] Interconnect resistance may be ignored without introducing much error. In addition, the transmission line properties of interconnect lines are ignored for on-chip connections.

rectangle approximation model, and the single trunk Steiner tree model. We use the last model as it is more accurate, yet can be computed efficiently. A single trunk Steiner tree consists of a single horizontal (vertical) trunk to which all nodes are joined by short vertical (horizontal) line segments. In order to compute the wire length, first the direction of the trunk is determined by considering the x and y direction spans of all the locations for nodes and picking the direction with larger total span. Then the location of the trunk is found by finding the median of the locations for nodes in the appropriate direction.

It is desirable to incrementally update the position of matched gates while the delay functions are being calculated. This operation results in positions which more accurately reflect the position of the gates after the mapping procedure. For this purpose, once a gate g is mapped at a node n, the position of g is updated by placing g at the center of the positions for its (mapped) fanin gates and fanout nodes. Each point on the area-delay curve of a node is thereby annotated with the position of the gate matching at the node.

The network is then mapped one logic cone at a time during the preorder traversal as described in section 3.2.

7 Experimental Results

These procedures have been implemented in a C program called ADmap. We have run the recommended set of the MCNC benchmarks [27] (except for C6288 benchmark where the detailed routing step aborts) using the ADmap and compared the results to the MIS2.2 technology mapper [24]. The same technology independently optimized *blif* files were used as input in both cases. The circuits were first optimized using the *script.rugged* [21]. They were then decomposed into NAND gates and mapped using the MIS2.2 [24] and the ADmap. Finally, the circuits were placed using GORDIANL [22] plus DOMINO [8] and routed using YACR [17]. All results are reported after layout is completed. We used the *lib2* library of the MIS2.2 package, assumed a value of $3pF/cm$ for the interconnect capacitance per unit length, and allowed a maximum of 16 points on each delay curve (see section 5).

Table 1 presents the total gate area and the longest path delay after technology mapping and the total chip area (including gate and wire areas) and the circuit delay (including gate and wire delays) in the area mode of MIS2.2 mapper (*map -s*). Table 2 shows results of the ADmap in the area mode: All entries in this table are normalized with respect to the corresponding entries in Table 1. For example, the post-mapping results of the ADmap for the *9symml* circuit are

$.1533 \times .9848 = .1510 mm^2$ and $24.07 \times .9238 = 22.23 ns$, respectively while its post-layout results are $.5697 \times 1.027 = 0.5851 mm^2$ and $31.82 \times .9484 = 30.18 ns$, respectively. On average, the ADmap produces post-mapping results with 5% less area and 4% less delay. The ADmap produces post-layout results with the same area, but with 3% less delay.

Table 3 shows the post-mapping and post-layout results for the MIS2.2 mapper in the timing mode (*map -n 1 -s*) while Table 4 contains the normalized results for the ADmap.[2] Entries in this table are normalized with respect to the corresponding entries in Table 3. On average, the ADmap produces circuits with 17% less area and 18% less delay (after mapping) or alternatively with 10% less area and 17% less delay (after layout). Table 5 shows the ADmap results for a different input parameter C (increasing C tends to increase area and reduce delay). In this case, the ADmap produces circuits with 4% less area and 28% less delay (after mapping) or alternatively with 8% more area, but 26% less delay (after layout).

The normalized placement-driven ADmap (PLmap) results are shown in Table 6 for the same input parameter as that used for generating the data in Table 5. Entries in this table are again normalized with respect to the corresponding entries in Table 3. The PLmap produces circuits with 12% less area and 24% less delay (after mapping) or alternatively with 4% less area and 22% less delay (after layout).

Table 7 contains the CPU time spent on a Sparc Station II with 64 MByte of memory for each mapper. The ADmap is on average 6.5 times slower than the MIS2.2 mapper while the PLmap is only 27% slower than the ADmap. The increase in run-time for the *C432* benchmark with 136 gates is 7.5, for the *C5315* benchmark with 1138 gates is 5.5, and for the *des* benchmark with 2880 gates is 8.5. This shows an almost constant increase in the run time of the ADmap over that of the MIS2.2 mapper. This result is reasonable in light of the complexity analysis given in section 5.

The actual post-layout % delay improvement of the ADmap over the MIS2.2 mapper is reduced if we do fanout optimization (Fanopt) after technology mapping. This is because circuits mapped by the MIS2.2 mapper have worse timing to begin with, thus the effect of fanout optimization on these circuits is more pronounced. Although the difference between ADmap and MIS2.2 mapper diminishes after this postprocessing step, the ADmap continues to outperform the MIS2.2 mapper by a good margin. This is demonstrated in Tables 8 and 9 which were generated as follows. After technology mapping by either ADmap or MIS2.2 mapper, we performed fanout optimization using the “-AFG” option of the MIS2.2 mapper which does fanout optimization followed by area recovery

[2]These results were generated without fanout optimization as we wanted to compare the technology mapping procedures without influence from other optimization procedures.

and a second fanout optimization pass (see [24] for details). The resulting circuits are then placed and routed.

Table 8 presents the total gate area and the longest path delay after technology mapping and fanout optimization, and the total chip area (including gate and wire areas) and the circuit delay (including gate and wire delays) in the timing mode of MIS2.2 mapper (*map -n 1 -AFG -s*). Table 9 shows results of the ADmap ($C = 1.0$) in the timing mode; As before, all entries in this table are normalized with respect to the corresponding entries in Table 8. On average, the ADmap with Fanopt produces circuits with 5% less area and 9% less delay (after fanout optimization) or alternatively with 8% less area and 11% less delay (after layout) compared with the MIS2.2 mapper with Fanopt. Similar results are obtained when comparing PLmap and MIS2.2 mapper.

Example	MIS2.2 (Area Mode)			
	Post-Mapping		Post-Layout	
	cell_area mm^2	delay ns	chip_area mm^2	delay ns
9symml	.1535	24.07	.5697	31.82
C1355	.3869	24.27	1.210	30.67
C1908	.4370	36.97	1.718	50.77
C2670	.6078	32.55	3.794	46.89
C3540	1.065	55.77	5.132	80.36
C432	.1958	45.74	.7119	60.46
C5315	1.408	38.96	8.735	56.45
C7552	1.866	82.67	11.97	122.3
C880	.3475	41.28	1.224	54.11
apex6	.6217	24.72	2.794	36.44
b9	.1127	8.71	.3805	11.50
dalu	1.054	68.81	4.713	102.8
des	2.756	177.3	16.60	295.9
k2	1.071	32.86	5.748	54.50
rot	.5860	26.86	3.149	38.98
t481	.6570	27.74	3.044	43.72

Table 1: MIS2.2 results in area mode

Example	ADmap (Area Mode)			
	Post-Mapping		Post-Layout	
	cell_area mm^2	delay ns	chip_area mm^2	delay ns
9symml	.9848	.9238	1.027	.9484
C1355	.9304	.9705	.8526	1.021
C1908	1.003	.9635	.9420	.9328
C2670	.9877	.9940	.9700	1.048
C3540	1.005	1.073	.9750	1.100
C432	1.018	1.075	.9955	1.092
C5315	1.004	1.060	1.001	1.054
C7552	1.006	.7881	.9854	.7556
C880	1.045	1.225	1.028	1.171
apex6	1.004	.7687	1.035	.7749
b9	1.049	.8250	1.053	.8773
dalu	1.040	1.029	1.052	1.023
des	1.023	1.107	.9946	1.073
k2	1.034	.8606	1.095	.8998
rot	1.022	.9667	1.021	.9645
t481	1.029	.7559	.9705	.7124
Average	.9514	.9615	.9998	.9654

Table 2: Normalized ADmap results in area mode

Example	MIS2.2 (Timing Mode)			
	Post-Mapping		Post-Layout	
	cell_area mm^2	delay ns	chip_area mm^2	delay ns
9symml	.2120	21.41	.7030	28.49
C1355	.5540	22.81	1.965	34.13
C1908	.5470	35.58	2.034	50.05
C2670	.8430	29.56	5.150	45.35
C3540	1.382	59.20	5.842	84.94
C432	.2612	43.38	.8713	56.03
C5315	1.744	38.64	10.62	52.14
C7552	2.719	66.02	15.17	96.54
C880	.4199	45.60	1.555	59.45
apex6	.8463	24.50	3.477	37.87
b9	.1364	9.140	.4193	11.58
dalu	1.568	62.55	5.986	92.96
des	4.251	226.5	22.38	317.4
k2	1.280	26.72	6.224	47.55
rot	.7205	27.33	4.165	42.20
t481	.8491	21.07	3.554	35.35

Table 3: MIS2.2 results in timing mode

Example	ADmap (Timing Mode))			
	Post-Mapping		Post-Layout	
	cell_area mm^2	delay ns	chip_area mm^2	delay ns
9symml	.7177	1.038	.8104	1.033
C1355	.7286	1.003	.6196	.8801
C1908	.9940	.8845	1.007	.8765
C2670	.7220	.8912	.7249	.8432
C3540	1.073	.8782	1.265	.9020
C432	.9662	.6774	1.060	.7146
C5315	.8659	.8571	.8912	.9672
C7552	.7221	.8250	.8231	.8055
C880	.8961	.8628	.8508	.8548
apex6	.7406	.7102	.7911	.7090
b9	.8673	.7861	.9703	.8298
dalu	.8689	.8031	1.093	.8299
des	.6672	.2093	.7548	.4677
k2	.8623	.8381	.9981	.8315
rot	.8628	.8688	.9553	.8656
t481	.7967	.9952	.8240	.8783
Average	.8343	.8204	.9017	.8304

Table 4: Normalized ADmap results in timing mode ($C = 0.5$)

Example	ADmap (Timing Mode))			
	Post-Mapping		Post-Layout	
	cell_area mm^2	delay ns	chip_area mm^2	delay ns
9symml	.9846	.7116	1.040	.7335
C1355	.9179	.8806	.8243	.7831
C1908	1.055	.8889	1.142	.9192
C2670	.8079	.6525	.8455	.6374
C3540	1.075	.8778	1.269	.9019
C432	1.420	.6518	1.842	.7404
C5315	.9010	.8239	.9305	.9367
C7552	.7899	.8211	.8854	.8083
C880	1.080	.6586	1.059	.6536
apex6	.7587	.6983	.8529	.6477
b9	1.020	.5955	1.221	.6545
dalu	.8808	.7848	1.079	.8169
des	.7918	.2653	.9196	.3341
k2	1.009	.6732	1.357	.8021
rot	.9079	.7353	1.010	.7011
t481	.9065	.8406	.9714	.8093
Average	.9561	.7224	1.077	.7424

Table 5: Normalized ADmap results in timing mode ($C = 1.0$)

Example	PLmap (Timing Mode))			
	Post-Mapping		Post-Layout	
	cell_area mm^2	delay ns	chip_area mm^2	delay ns
9symml	.8687	.7403	.9483	.8122
C1355	.8283	.9007	.7415	.7972
C1908	1.033	.8875	1.130	.9014
C2670	.7303	.7805	.7335	.7847
C3540	1.039	.8411	1.233	.9254
C432	1.053	.7025	1.243	.7399
C5315	.9582	.8954	.9645	1.036
C7552	.7356	.8071	.8284	.8008
C880	.9337	.7501	.9301	.7547
apex6	.7538	.7102	.8448	.6683
b9	1.040	.5499	1.213	.5518
dalu	.8107	.9354	.9332	.9339
des	.6906	.3003	.7928	.3669
k2	.9010	.7272	1.048	.7432
rot	.9175	.7651	1.027	.7310
t481	.8087	.9164	.8380	.8724
Average	.8813	.7630	.9646	.7762

Table 6: Normalized PLmap results in timing mode ($C = 1.0$)

Example	MIS2.2 map	ADmap	PLmap
	s	s	s
9symml	9.4	35.3	44.4
C1355	21.9	127.1	159.7
C1908	21.5	180.1	213.4
C2670	32.2	176.6	222.0
C3540	51.1	524.2	616.5
C432	11.1	83.0	100.3
C5315	66.4	365.5	480.2
C7552	85.7	663.3	796.7
C880	18.0	81.4	105.5
apex6	31.6	75.2	116.2
b9	5.8	6.1	9.8
dalu	52.6	587.0	708.3
des	130.5	1107.1	1245.8
k2	46.3	357.5	433.9
rot	28.1	84.0	122.5
t481	34.0	340.4	359.9

Table 7: CPU times on a Sparc Station II with 64 MByte of memory

Example	MIS2.2 (Timing Mode)					
	Post-Mapping		Post-Fanopt		Post-Layout	
	cell_area mm^2	delay ns	cell_area mm^2	delay ns	chip_area mm^2	delay ns
9symml	.2120	21.41	.2213	15.56	.7762	20.82
C1355	.5540	22.81	.6296	19.68	2.183	29.93
C1908	.5470	35.58	.5521	27.85	2.134	41.67
C2670	.8430	29.56	.7971	20.12	5.167	35.15
C3540	1.382	59.20	1.353	39.98	6.000	64.22
C432	.2612	43.38	.2677	26.37	.9171	34.92
C5315	1.744	38.64	1.698	26.90	10.32	41.35
C7552	2.719	66.02	2.334	35.21	14.98	60.34
C880	.4199	45.60	.4315	31.91	1.635	42.65
apex6	.8463	24.50	.8124	16.79	3.517	28.66
b9	.1364	9.140	.1350	7.480	.4295	9.830
dalu	1.568	62.55	1.462	41.12	6.062	61.56
des	4.251	226.5	3.660	22.84	21.18	47.40
k2	1.280	26.72	1.276	18.02	6.815	36.21
rot	.7205	27.33	.7057	17.67	3.649	27.72
t481	.8491	21.07	.8607	15.53	3.818	27.24

Table 8: MIS2.2 results in timing mode

Example	ADmap (Timing Mode)					
	Post-Mapping		Post-Fanopt		Post-Layout	
	cell_area mm^2	delay ns	cell_area mm^2	delay ns	chip_area mm^2	delay ns
9symml	.9846	.7116	.8280	.9214	.8458	.9046
C1355	.9179	.8806	.8372	.9157	.7940	.9291
C1908	1.055	.8889	.9563	1.017	.8652	1.003
C2670	.8079	.6525	0.940	.8020	.8178	.7638
C3540	1.075	.8778	1.050	.8939	1.034	.8873
C432	1.420	.6518	1.137	.9351	1.080	.9050
C5315	.9010	.8239	.8679	.9676	.8099	.9124
C7552	.7899	.8211	.9791	1.008	.7868	.8810
C880	1.080	.6586	1.023	.7972	1.090	.8186
apex6	.7587	.6983	.8817	.8600	.8583	.8384
b9	1.020	.5955	1.054	.8677	1.077	.8921
dalu	.8808	.7848	.8883	1.029	.9493	1.015
des	.7918	.2653	.9425	1.031	.9781	1.012
k2	1.009	.6732	1.035	.8485	1.037	.9022
rot	.9079	.7353	.9240	.8519	.8282	.7892
t481	.9065	.8406	.9417	.8613	.9050	.8377
Average	.9561	.7224	.9515	.9129	.9222	.8932

Table 9: Normalized ADmap results in timing mode ($C = 1.0$)

Figure 4 goes here.

Figure 5 goes here.

The graphs in Figures 4 and 5 show a range of mapped solutions produced for two benchmark circuits. These plots serve to illustrate the generality of our method in obtaining a range of solutions with different tradeoffs under user control. Our algorithm subsumes technology mapping techniques for minimum area and/or delay.

8 Conclusions

We have presented a powerful technique for technology mapping which generates solutions with different area/delay tradeoffs. Our technique unifies techniques for technology mapping with different objectives (minimum area, maximum performance, and minimum area under delay constraints) and is based on principles of dynamic programming and computation of delay curves. For a node-balanced NAND-decomposed tree, our algorithm finds the optimum area solution under delay constraints (subject to error due to unknown loads during delay computation step) in polynomial time and space. For the general problem of mapping DAGs, the algorithm retains its efficiency and produces results which are superior to those produced by other mappers.

The ADmap program has been extended to generating circuits with minimum average power consumption under delay constraints by constructing optimal (under a non-glitch delay model) power-delay curves during the postorder phase [25]. Extension to combine pin permutation with technology mapping is straight-forward, but increases the complexity of the procedure.

References

[1] H. B. Bakoglu. *Circuits, interconnections, and packaging for VLSI*. Addison-Wesley, 1990.

[2] K. Bartlett, W. Cogen, A. de Geus, and G. Hachtel. Synthesis and optimization of multilevel logic under timing constraints. IEEE *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-5(4):582–596, October 1986.

[3] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. In *Proceedings of the* IEEE, volume 78, pages 264–300, February 1990.

[4] K. Chaudhary and M. Pedram. A near-optimal algorithm for technology mapping minimizing area under delay constraints. Proceedings of the 29th Design Automation Conference, June 1992.

[5] H. Chen and D. H. C. Du. Path sensitization in critical path problem. IEEE *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 196–207, February 1993.

[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):537–545, September 1984.

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

[8] K. Doll, F. M. Johannes, and G. Sigl. DOMINO: Deterministic placement improvement with hill-climbing capabilities. In *IFIP International Conference on Very Large Scale Integration*, 1991.

[9] Y. A. El-Mansy and W. M. Siu. MOS technology advances. In G. Rabbat, editor, *Handbook of Advanced Semiconductor Technology and Computer Systems*, pages 229–259. Van Nostrand-Reinhold, Princeton, New Jersey, 1988.

[10] K. Keutzer. DAGON: Technology mapping and local optimization. In *Proceedings of the Design Automation Conference*, pages 341–347, June 1987.

[11] R. Lisanke, F. Brglez, and G. Kedem. Mcmap: A fast technology mapping procedure for multi-level logic synthesis. In *Proceedings of the International Conference on Computer Design*, pages 252–256, October 1988.

[12] F. Mailhot and G. D. Micheli. Technology mapping using boolean matching. In *Proceedings of the European Design Automation Conference*, pages 180–185, March 1990.

[13] P. McGeer and R. Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *Proceedings of the 26th Design Automation Conference,* pages 561–567, June 1989.

[14] C. R. Morrison, R. M. Jacoby, and G. D. Hachtel. TECHMAP: Technology mapping with delay and area optimization. In *Proceedings of the International Workshop on Logic and Architecture Synthesis for Silicon Compilers*, pages 53–64, Grenoble, France, May 1988.

[15] M. Pedram and N. Bhat. Layout driven technology mapping. In *Proceedings of the 28th Design Automation Conference*, pages 99–105, June 1991.

[16] M. Pedram and H. Vaishnav. Technology decomposition using optimal alphabetic trees. In *Proc. European Conf. on Design Automation*, pages 573–577, 1993.

[17] J. Reed, A. Sangiovanni-Vincentelli, and M. Santamauro. A new symbolic channel router: YACR2. *IEEE Trans. on Computer-Aided Design*, 4(3):208–219, March 1985.

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

[19] T. Sakurai. Approximation of wiring delays in MOSFET LSI. IEEE *Journal of Solid State Circuits*, SC-18(4):418–426, August 1983.

[20] K. C. Saraswat and F. Mohammadi. Effect of scaling of interconnections on the time delay of VLSI circuits. IEEE *Transactions on Electron Devices*, ED-29:645–650, 1982.

[21] H. Savoj and H. Y. Wang. Improved scripts in MIS-II for logic minimization of combinational circuits. In *Proceedings of the International Workshop on Logic Synthesis*, May 1991.

[22] G. Sigl, K. Doll, and F. M. Johannes. Analytical placement: A linear or a quadratic objective function? In *Proceedings of the 28th Design Automation Conference*, pages 427–432, June 1991.

[23] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 57:91–101, 1983.

[24] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang. Performance-oriented technology mapping. In *Proceedings of the Sixth M.I.T. Conference on Advanced Research in VLSI*, pages 79–97, April 1990.

[25] C-Y. Tsui, M. Pedram, and A. M. Despain. Technology decomposition and mapping targeting low power dissipation. In *Proceedings of the 30th Design Automation Conference*, pages 68–73, June 1993.

[26] A. Wang. *Algorithms for multi-level logic optimization*. PhD thesis, University of California, Berkeley, April 1989. Technical Report UCB/ERL M89/50.

[27] S. Yang, editor. *Logic Synthesis and Optimization Benchmarks User Guide*. Microelectronics Center of North Carolina, Research Triangle Park, North Carolina, 1991.

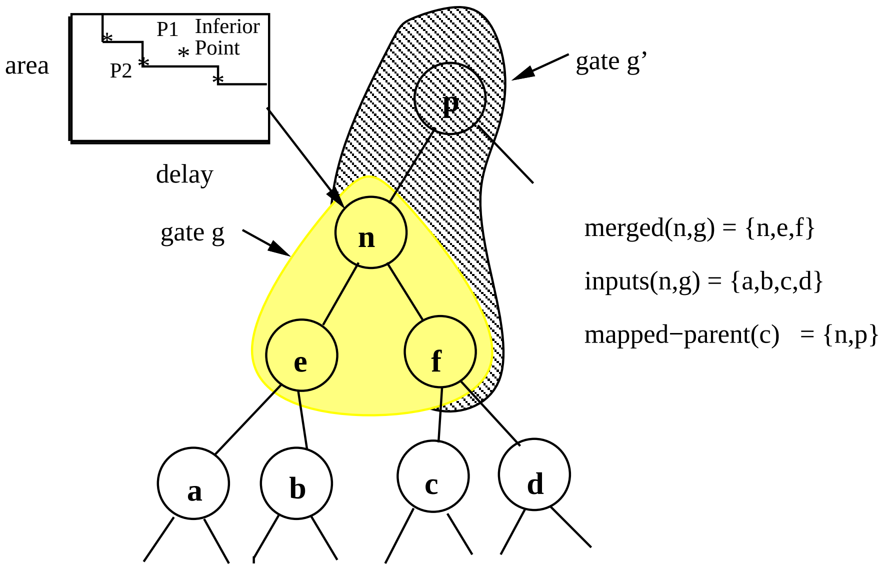

Figure 1: Terminology

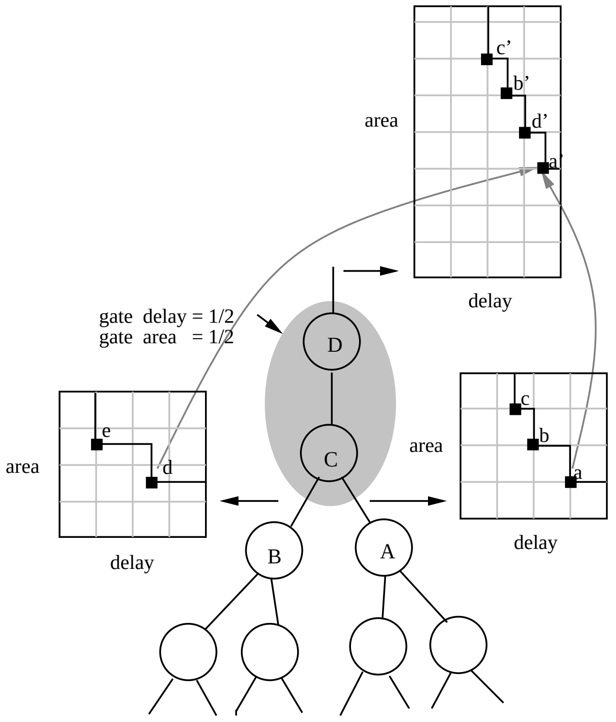

Figure 2: Generating the delay curve for a given match

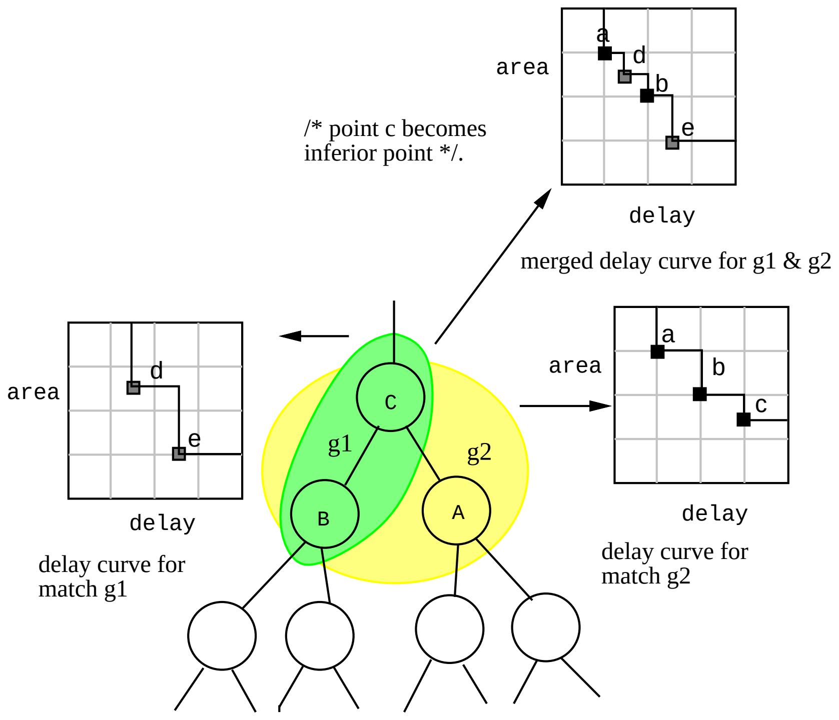

Figure 3: Lower bound merging of delay curves

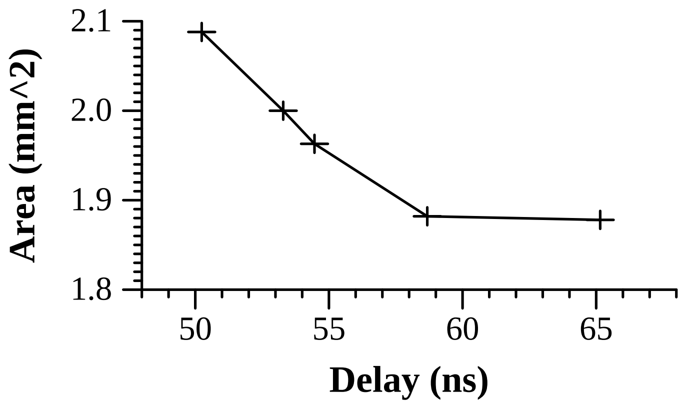

Figure 4: The delay curve for the *C7552* benchmark circuit

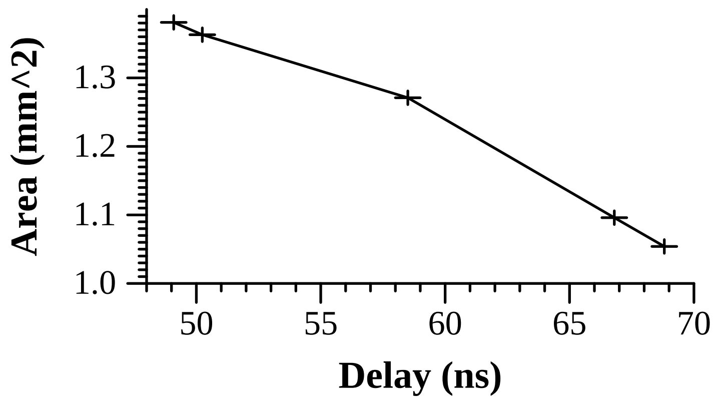

Figure 5: The delay curve for the *dalu* benchmark circuit