A Near Optimal Algorithm for Technology Mapping Minimizing Area under Delay Constraints

Kamal Chaudhary
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

Massoud Pedram
Department of Electrical Engineering – Systems
University of Southern California, Los Angeles, CA 90089

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 arrival time – gate area 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 all delay constraints 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). 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.

1.1 Problem Definition

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 in [3] and implemented in programs such as DAGON and MIS. 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. [9] extended this approach to solve the technology mapping problem minimizing delay (subject to error due to unknown load values during mapping) and the technology mapping problem minimizing area under delay constraints. The solution to the latter problem, however, is not satisfactory.

The procedure for converting an optimized Boolean network into a two-input NAND and inverter decomposed (for short, NAND-decomposed) DAG is not unique and it is an open problem to determine which of the possible subject DAGs yields an optimum solution when an optimum covering algorithm is applied [2]. Different decomposition schemes for minimizing area [6], minimizing delay [11], or reducing routing complexity [5] 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.

1.2 Prior Work

In [1], the author gives a pseudo-polynomial time algorithm for computing the min-max slacks (given the minimum and maximum required times at the primary outouts) at all internal nodes of a tree. The technique, however, can be used only for gate sizing (i.e., the tree has already been mapped, the problem is to assign the optimal size to each gate from a

discrete number of sizes in the target library while satisfying the timing constraints).

In [9], 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 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.

1.3 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. We present our results and concluding remarks in Sections 6 and 7.

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 which 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 some node n for which there exists a matching gate g such that $n_i \in inputs(n, g)$. Note that given node n and gate g matching at n, $inputs(n, g)$ are uniquely determined. However, n_i may have many distinct mapped parents (Figure 2.1).

With each node in the network we store a delay curve. 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

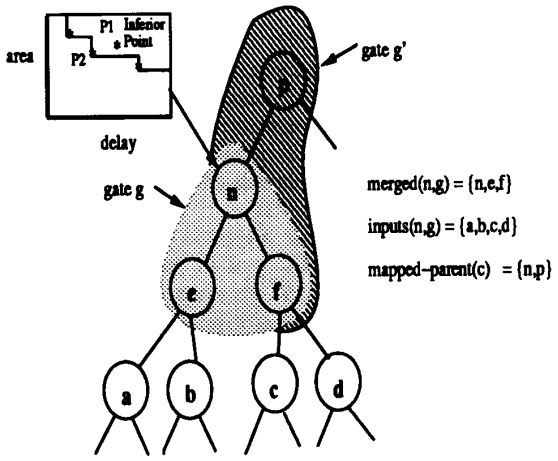

Figure 2.1: Terminology

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.

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

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

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

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 [8] 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 in order to ensure that the resulting 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.

To illustrate the delay function computation procedure, consider example in Figure 3.1. It shows the computation

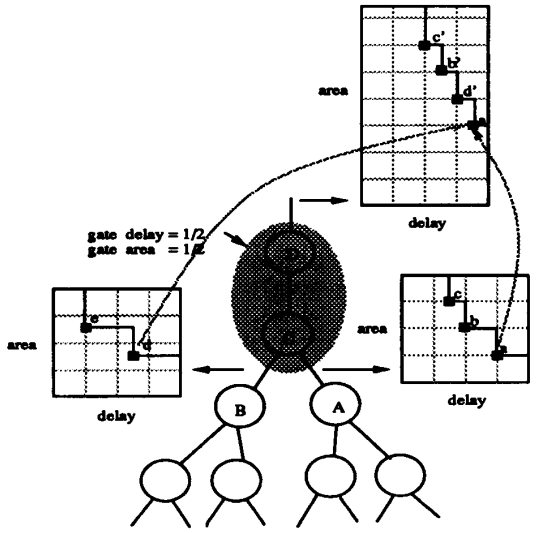

Figure 3.1: Generating the delay curve for a given match

of 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).

To illustrate the lower bound merging procedure, consider example in Figure 3.2. Here, we have already generated the delay-curves for the matching gates g_1 and g_2 at node n. 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.

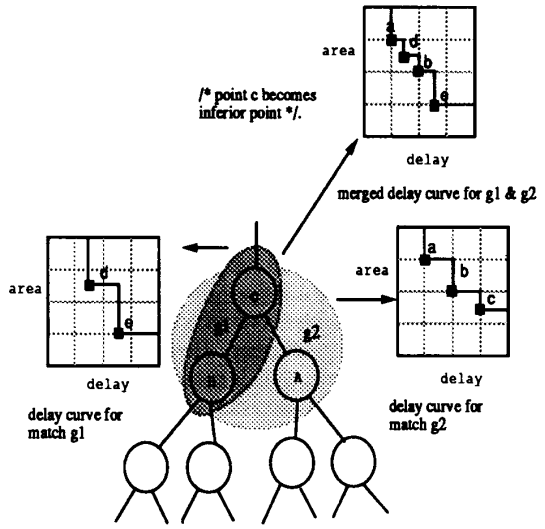

Figure 3.2: Lower bound merging of delay curves

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 which 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). At this time the load value is assumed to be that offered by the smallest two-input NAND gate in the library. When we come to a node n =*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 [4]. 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 ($_i$), 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 $_i \times p_j.gate.drive$. (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
      Γ_i = g.intrinsic_i + g.drive_i × n.load
      Δ_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 + Δ_i × p_j.gate.drive + Γ_i
        feasible-flag = true
        for each input k of match g do
          if k = i, continue
          find p_k* where p_k*.area is minimum and
               p_k*.delay ≤ np.delay − _k
          if no such p_k* found
            feasible-flag = false
            break;
          end
        end
        if feasible-flag = true
          np.gate = g
          np.binding = inputs(n, g)
          np.area = g.area + Σ_k p_k*.area
          insert np to 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* on delay-curve(n) which satisfies the
    required time t and has minimum area
  n.best_gate = p*.gate
  n.best_binding = p*.binding
  for each input i ∈ inputs(p*.gate, n) do
    _i = g.intrinsic_i + g.drive_i × n.load
    assign_best_gate(n_i, t − _i)
  end
end
```

3.4 Taking the Load Values into Account

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 it is $\leq \epsilon$, then no significant loss of optimality. ∎

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 fanout count, the expected average interconnect length and 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 [2]. 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 only for nodes with relatively few fanouts was also adopted by the MIS 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.

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. It will require another $Nlog(N)$ to sort the

N points. Thus, the total time for generating delay curve per candidate match is $O(N^2 log(N) k log(N_m))$. A more efficient implementation based on merging of sorted lists will have $O(N)$ run time.

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 $O(k^2 P^2 log(kP) log(P))$ or $O(k^2 log(k))$, which is a constant since the number of inputs for any gate in the library is bounded.

6 Experimental Results

The procedure has been implemented in a computer program named ADIEU. We have run the MCNC benchmarks using ADIEU and compared the results to the MIS2.2 technology mapper [9]. The same technology independent optimized *blif* files were used as input in both cases. The circuits were first optimized using *script.rugged* [7] and then delay optimized using the MIS2.2 new delay script [10]. Finally, they were decomposed into NAND gates. We mapped the circuits using the MIS2.2 and ADIEU mappers (without fanout optimization). We used the *lib2* library of the MIS2.2 package.

Table 1 presents the total gate area and the longest path delay through the circuit in the area mode of ADIEU. In Tables 2 and 3, all numbers have been normalized to the area mode of ADIEU. On average, ADIEU's area mode produces circuits which are faster by 6%, but larger by 3% (compared to MIS2.2's area mode); ADIEU's timing mode produces circuits as fast as MIS2.2's timing mode, but with 17% less area.

The graph in Figure 7.1 shows a range of mapped solutions produced for *C432* benchmark when using with different cycle times. These plots serve to illustrate the universality of our method in obtaining a range of solutions with different tradeoffs under user control. Our algorithm subsumes technology mapping techniques for minimum area or maximum performance.

Example	ADIEU	
	Area mode	
	area $*10^4$ μ^2	delay (ns)
9symml	15.5	18.8
apex6	76.4	37.0
apex7	25.5	16.2
b9	12.8	8.7
des	329.7	66.5
rot	68.9	23.4
z4ml	4.1	11.0
C1908	61.8	36.2
C1355	55.4	23.8
C432	25.9	33.7
C880	43.4	36.4
C3540	118.5	49.2
C5315	168.1	39.6
C7552	247.8	86.9

Table 1: Technology mapping results

7 Concluding Remarks

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.

We plan to combine this technique with the layout driven mapping technique of [4] in order to include and improve the wire load and routing area estimation during generation of the delay curve and gate selection.

Acknowledgements

The authors would like to thank Professor Ernest S. Kuh for his support and encouragement and Narasimha Bhat for valuable discussions. This research was supported in part by the National Science Foundation under grant number MIP 88-03711 and by the Defense Advanced Research Projects Agency under contract number JFBI90092.

References

[1] P. K. Chan, "Algorithms for library-specific sizing of combinational logic," *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 352-356, 1990.

Example	MIS2.2			
	Area mode		Timing mode	
	area	delay	area	delay
9symml	1.00	0.99	1.35	0.91
apex6	0.94	0.92	1.35	0.90
apex7	0.97	0.97	1.26	0.88
b9	0.96	0.93	1.06	0.78
des	0.98	2.19	1.43	1.24
rot	0.99	0.98	1.18	1.02
z4ml	0.93	1.01	1.24	0.87
C1908	0.93	1.16	1.27	1.13
C1355	0.97	0.92	1.18	0.86
C432	1.00	1.00	1.26	0.81
C880	0.98	0.99	1.17	0.83
C3540	0.99	1.00	1.27	0.96
C5315	0.98	0.89	1.28	0.84
C7552	0.97	1.03	1.31	0.74
average	0.97	1.06	1.26	0.91

Table 2: Normalized mapping results for MIS2.2

Example	ADIEU			
	Area mode		Timing mode	
	area	delay	area	delay
9symml	1.00	1.00	1.09	0.90
apex6	1.00	1.00	1.03	0.75
apex7	1.00	1.00	1.01	0.93
des	1.00	1.00	1.00	0.89
b9	1.00	1.00	1.06	0.92
rot	1.00	1.00	1.01	0.95
z4ml	1.00	1.00	1.10	0.87
C1908	1.00	1.00	1.04	1.01
C1355	1.00	1.00	1.10	0.93
C432	1.00	1.00	1.07	0.83
C880	1.00	1.00	1.07	0.90
C3540	1.00	1.00	1.04	0.91
C5315	1.00	1.00	1.05	0.85
C7552	1.00	1.00	1.03	0.90
average	1.00	1.00	1.05	0.91

Table 3: Normalized mapping results for ADIEU

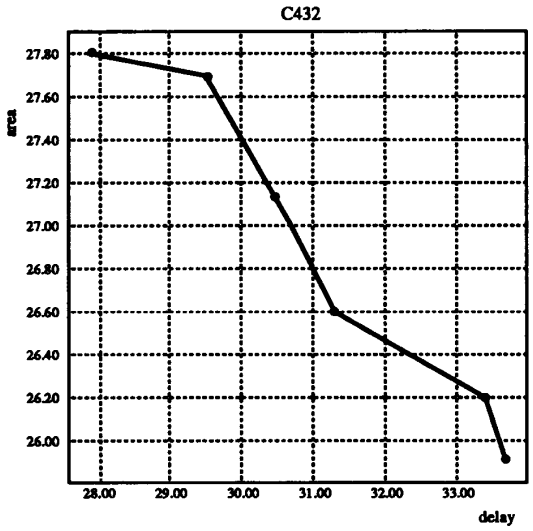

Figure 7.1: Delay-curves for *C432* benchmark

[2] R. K. Brayton, G. D. Hachtel and A. L. Sangiovanni-Vincenntelli, "Multilevel logic synthesis," *Proc. of the IEEE*, vol 78, no. 2, pp. 264-300, February 1990.

[3] K. Keutzer, "DAGON: technology binding and local optimization by DAG matching," *Proc. 24th ACM/IEEE Design Automation Conference*, pp. 341-347, 1987.

[4] M. Pedram and N. Bhat, "Layout driven technology mapping," *Proc. 28th ACM/IEEE Design Automation Conf.*, pages 99-105, 1991.

[5] M. Pedram and N. Bhat, "Layout driven logic restructuring / decomposition," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 134-137, 1991.

[6] R. Rudell, "Logic synthesis for VLSI design," Ph.D. dissertation, University of California, Berkeley, April 1989.

[7] H. Savoj, H. -Y. Wang, "Improved scripts in MIS-II for logic minimization of combinational circuits," *Proc. Int. Workshop on Logic Synthesis*, Vol. 3, 1991.

[8] L. Stockmeyer, "Optimal orientation of cells in slicing floorplan designs," *Information and Control*, Vol. 57, pages 91-101, 1983.

[9] H. J. Touati, C. W. Moon, R. K. Brayton and A. Wang, "Performance-oriented technology mapping," *Proc. 6th MIT Conf, Advanced Research in VLSI*, W. J. Dally ed., pp. 79-97, 1990.

[10] H. J. Touati, H. Savoj, and R. K. Brayton, "Delay optimization of combinational logic circuits by clustering and partial collapsing," *Proc. IEEE Int. Conf. Computer Design*, pages 188-191, 1991.

[11] A. Wang, "Algorithms for multi-level logic optimization," Ph.D. dissertation, University of California, Berkeley, April 1989.