Layout Driven Technology Mapping

Massoud Pedram and Narasimha Bhat
Department of Electrical Engineering and Computer Science
University of California, Berkeley CA 94720

Abstract

Recent studies indicate that interconnections occupy more than half the total chip area and account for a significant part of the chip delay. In spite of this, most logic synthesis systems do not explicitly take the wiring into account during the optimization phase. Our work is a first step towards including wiring into the logic synthesis process. In this paper, we present *Lily*, a technology mapper integrated with MIS, which considers layout area and wire delay during the technology dependent phase of logic synthesis. *Lily* estimates the interconnection dependent contributions to circuit area and delay by referring to a dynamically updated global placement of the Boolean network. The update does not restrict the dynamic programming approach adopted in technology mappers such as DAGON and MIS. Our algorithm has been implemented and preliminary results are encouraging.

1 Introduction and Motivation

The goal of logic synthesis is to produce a circuit which satisfies a set of logic equations, occupies minimal silicon 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 [11, 10]. 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. It is assumed that wiring optimization can be handled efficiently in the physical design phase.

Decisions made during the logic synthesis phase may limit the optimization potential of physical design tools. For example, excessive factorization based on common kernel extraction during the technology independent phase of logic synthesis can lead to gates with high fanout count and increased path delay. Inordinate attention has been focused on minimizing the active cell area, during technology mapping, leading to gates with high fanin count which often increase routing congestion during the final layout and increase interconnection lengths. Ignoring propagation delay through wires has introduced inaccuracy in the timing analysis performed during technology mapping.

With recent studies [13] 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 we integrate wiring into the cost function for logic synthesis. The work presented in this paper is a step toward fusing layout considerations into the logic synthesis process. Specifically, we incorporate the wiring area and delay into the technology mapping phase.

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 the nodes in the network to gates in the library such that area of the final implementation (after gate placement and routing) is minimized and timing constraints are satisfied. A successful and efficient solution to this problem was suggested by Kurt Keutzer and implemented in DAGON [8] and MIS [9]. 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[2]. DAGON and MIS technology mappers generate circuits with small active cell area but ignore area and delay contributed by interconnections between gates.

We justify incorporating wiring estimates into technology mapping by pointing out problems associated with minimizing only active cell area. Figure 1.1a shows a small portion of a Boolean network. Source nodes s_i have either been mapped (and hence have been assigned matching gates and positions) or are fixed at the chip boundary. Note that s_1 and s_2 have positions near one another but are far from s_3 and s_4. Our objective is to transfer the signals from s_i's to the sink node t implementing the desired logic function while using minimum wire length. The decision problem can be stated as: "Is there a minimum wire length solution with the number of *distribution* points $\leq k$?"[1] Technology mappers such as DAGON and MIS attempt to find a solution with $k = 1$, i.e., they find the smallest area gate which matches as many intermediate nodes as possible. This is a good approach if the fanin gates, s_i, can be placed near the matching gates. However, in many cases, these gates are either strongly connected to different gate clusters on the layout plane or are fixed at the chip boundary and hence may have positions far from one another and from the matching gate. Therefore, a solution with one distribution point may incur a large interconnection cost. In fact, there is often an optimum $k > 1$ which will result in overall minimum wire cost as illustrated graphically in Figure 1.1a. Note that if the number of sources is small, say 3, one distribution point will suffice for achieving both minimum active cell area and minimum wire cost. However, if the number of sources is large, say 5 or more, then it will pay off to consider how close the sources can be placed by a good placement optimizer before deciding whether a solution of one gate (with high fanin count) or a solution of more than one gate (with low fanin counts) should be accepted during the technology mapping process.

[1] Here, a distribution point refers to a logic gate between the sources and the sink.

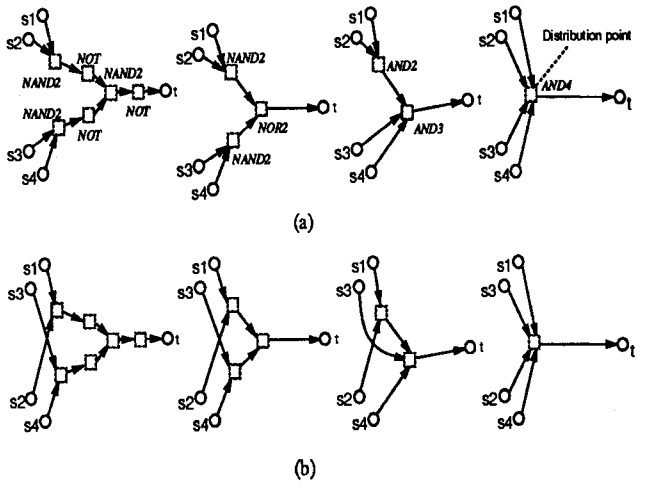

Figure 1.1: (a) Active gate area versus wire length (b) Motivation for a layout oriented decomposition

Figure 1.1b illustrates the importance of a good decomposition for the layout-driven technology mapping scheme. In this figure, we show the same decomposition tree as in Figure 1.1a. However, this time, as a result of placing the Boolean network or dynamic updating of node positions, source nodes s_1 and s_3 (s_2 and s_4) have been positioned near one another. Signals coming from s_1 and s_3 (s_2 and s_4) enter the decomposed network (subject graph, during technology mapping) at topologically distant points. This is undesirable because the decomposition tree conflicts with the placement solution (which reflects the global connectivity structure of the network). The mapper has lost the option of reducing the wiring cost by breaking one big match into smaller matches. Generalizing this observation, we seek a decomposition of the logic function associated with each node in the optimized Boolean network such that the fanin signals which are coming from nearby regions in the companion placement solution enter the decomposition tree at topologically near points. For example, Figure 1.1a provides a better decomposition (and hence potential for higher quality mapping) than that in Figure 1.1b. Note that this distinction does not arise if the mapper chooses to ignore wiring cost and to minimize only gate area.

In this paper, we present *Lily*, a technology mapper based on DAG covering which integrates gate placement and interconnection length estimation with the dynamic programming algorithm. *Lily* maps a given logic circuit onto a set of gates in the target library such that *layout* area and delay are minimized. The layout area is the sum of gate areas and routing area. The delay in the circuit is contributed by gates and interconnections among them. We estimate the interconnection dependent contributions to circuit area and delay by referring to a dynamically updated global placement of the Boolean network. This updating is consistent with the dynamic programming approach adopted in technology mappers such as DAGON and MIS.

The rest of the paper is organized as follows. In Section 2, we state the terminology and notations used throughout the paper. In Sections 3 and 4 we discuss technology mapping targeted towards area minimization and delay minimization, respectively. Experimental results and concluding remarks are presented in Sections 5 and 6.

2 Terminology

The DAG covering approach to technology mapping can be summarized as follows [16]. 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.) A *sink* node in a pattern graph is defined as a node which does not fanout to any other node in the pattern graph. 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, the cost of a cover is defined as the critical path delay of the resulting circuit.

Consider a Boolean network, N, which has been transformed into a subject graph consisting of only 2-input nand and inverter gates. This is the network in its unmapped form which we shall refer to as the *inchoate* network, $N_{inchoate}$. In DAGON, $N_{inchoate}$ is partitioned into a set of maximal *trees*, T_i, and an optimal dynamic programming solution is found for each tree. In MIS, $N_{inchoate}$ is split into a set of *logic cones*, K_i, where each cone corresponds to a primary output and all its transitive fanin nodes. This allows covering across tree boundaries and, as a result, may duplicate logic. The MIS technology mapper implements DAGON as a subset.

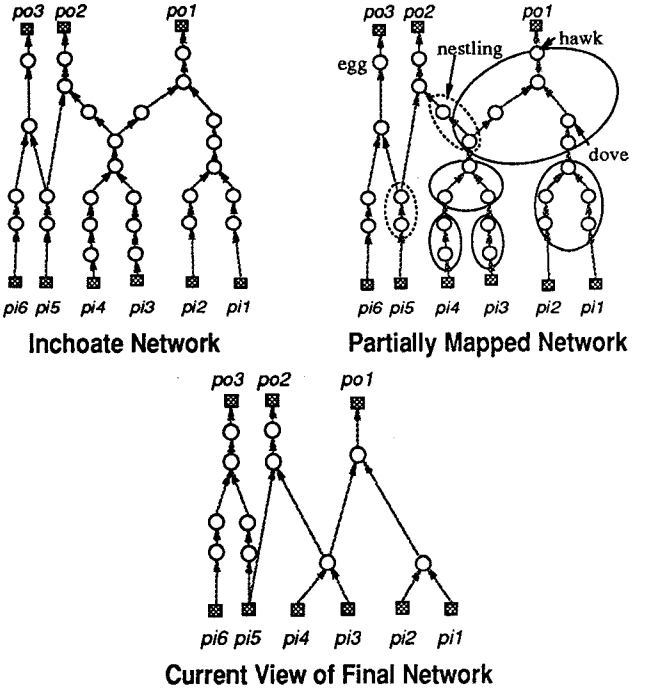

Figure 2.1: Incremental updating of the Boolean network

Consider Figure 2.1 which shows an example $N_{inchoate}$ at some point during the mapping process. Assume that we have processed cone K_1 corresponding to primary output po_1. We have also processed some of the nodes in cone K_2 and have to process the remaining nodes in cone K_2 as well as nodes in cone K_3. (In the dynamic programming approach, we start from the primary inputs of the logic cone and recursively process nodes in a reversed depth first search order toward the primary output.) At this point, nodes in $N_{inchoate}$ can be classified into four categories. An *egg* is a node which has not been processed (visited) by the mapper. A *nestling* is a node in the current cone, K_2, which has been visited. We cannot predict whether or not a *nestling* will be present in the final mapped

network, N_{mapped}, until we reach po_2. A *dove* is a node in K_1 which is a non-sink element of some pattern match. Such a node will not be present in N_{mapped} because it has been merged into another. A *hawk* is a node in K_1 which is a sink node in some pattern match. Such a node will inevitably show up in N_{mapped}. Note that every *dove* has been merged into (fallen prey to) at least one *hawk*. A *nestling* can become a *hawk* or a *dove*. Due to the possibility of logic duplication, it may be possible for a *dove* to reincarnate and restart the node's life cycle as an *egg* and later become a *hawk*. (See Figure 2.2.) At the end of the mapping procedure, only *hawks* and *doves* remain. This classification will be used later to describe the construction of fanin rectangles which are needed for updating placement positions and estimating wire lengths.

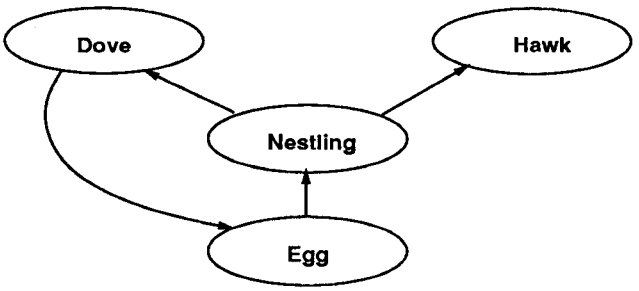

Figure 2.2: A node's life cycle during the mapping

A *stem* refers to a multiple-fanout node in $N_{inchoate}$. A *branch* is the immediate fanout node of a *stem*. A *line* refers to a directed edge in $N_{inchoate}$. An *exit line* for a cone K_i is a line which is an output line of a node in K_i and input line of a node which is not in K_i.

3 Technology Mapping for Minimum Layout Area

We intend to find a covering of a subject graph G by a set of pattern graphs P such that layout cost is minimized. The layout cost refers to the actual area of the implementation after placement and routing. *Lily*'s cost function accounts for the gate area and the routing area.

Assume that we are evaluating the cost of match m at node v. (See Figure 3.1.) This cost consists of two components:

$$aCost(v,m) = area(gate(m)) + \sum_{v_i} aCost(v_i)$$

$$wCost(v,m) = wire(gate(m), gate(v_i)) + \sum_{v_i} wCost(v_i)$$

Here, $v_i \in inputs(v,m)$, where $inputs(v,m)$ refers to the list of nodes of G which correspond to the inputs of m. $gate(m)$ is the physical gate corresponding to match m. $gate(v_i)$ is the best gate matching at node v_i. The area cost calculation is straight forward and is similar to that in MIS. The wire cost, $wCost(v,m)$, consists of two terms. The first term is the interconnection length required to complete connections from $gate(m)$ to its fanin gates, i.e., $gate(v_i)$. The latter is the dynamic programming recursive cost and represents the sum of wire lengths required to connect all gates from primary inputs up to $gate(v_i)$.

3.1 Global Placement

We use a global placement procedure [14, 21] to place the base function gates in $N_{inchoate}$ on a layout image. The actual area of the image is estimated by accurate area predictors for standard cell based designs such as that in [15]. Prior to the mapping process, positions of the I/O pins (primary inputs and outputs of the logic circuit) are assigned either by a top-down floorplanning and pin assignment procedure driven by system-level considerations (as in [19]) or by a bottom-up I/O pin assignment procedure driven by the connectivity structure of $N_{inchoate}$ (as in [20]).

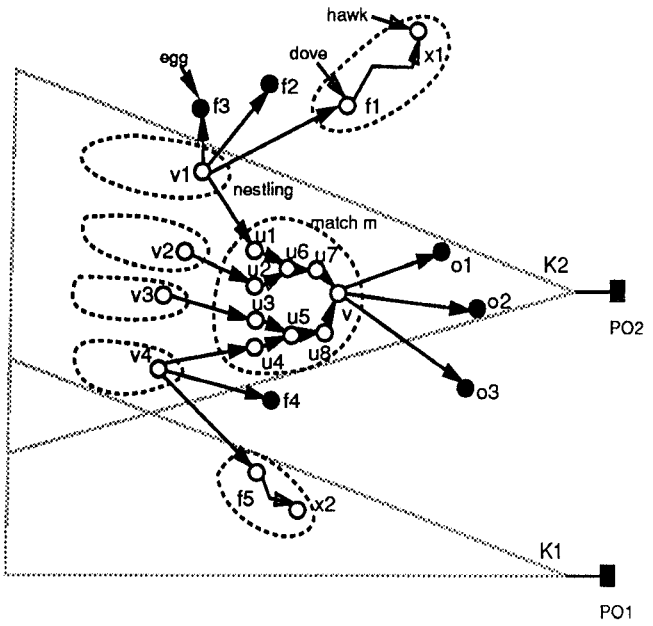

Figure 3.1: Cost calculation for a candidate match

The global placement phase generates a balanced point placement for all gates subject to the given I/O pad assignment which minimizes the Euclidean distance squared metric summed over all connected gates. It uses quadratic optimization and bi-partitioning techniques to place the gates. The bi-partitioning step may be stopped when the number of modules assigned to any subregion is less than some user-specified parameter. (A limit of one module per region and an assignment of modules to rows or slots corresponds to a detailed placement.) By a balanced global placement, we mean that the gates are uniformly distributed within the chip boundary, i.e., there are no over-subscribed or under-subscribed subregions.

Such a global placement is desirable for two reasons. Firstly, the incentive for the global placement is to capture the connectivity structure of the Boolean network on a plane. We do not want to destroy the global optimality of the solution and the ability to capture the logic structure by prematurely forcing gates into rows (which would be the case, if we did a detailed placement). Secondly, the placement updating procedure does not perform well on a two-dimensional mesh due to the inability to keep track of the slot capacity constraints during the dynamic programming process.

We use a point model during the gate placement. The gate pins are assumed to be located at the center of the gate and the location of the gate is represented by a single (x,y) coordinate that coincides with the center of the gate. These assumptions do not introduce much error when the number of gates in the circuit is large.

3.2 Incremental Updating of Placement

Initially, nodes in $N_{inchoate}$ are assigned valid *placePositions* based on the global placement solution. As nodes are mapped, new *mapPositions* are calculated and assigned to them. There are two options for computing the *mapPositions*. In the *CM-of-Merged* op-

tion, we place match m at the center of mass of $merged(v, m)$. (This is the list of nodes of $N_{inchoate}$, including v, which are 'covered by' or 'merged into' m.) The calculation uses *placePositions* for u_i. (See Figure 3.1.) In the *CM-of-Fans* option, we place match m such that the wire length to $inputs(v, m)$ and to $outputs(v)$ is minimized. Due to depth first search ordering, $inputs(v, m)$ have already been mapped and therefore we use their *mapPositions*. $outputs(v)$ are not mapped yet and we use their *placePositions*.

The advantage of the first approach is that the *mapPositions* are always calculated by referring to the global placement result. Since the initial placement is balanced and captures the adjacency relations among nodes in $N_{inchoate}$, the evolving placement will also be balanced. Note that mapping decisions made at node v are influenced by the estimated wire length cost which is computed as a function of the distances between $gate(m)$ and the best gates matching at $inputs(v, m)$ and between $gate(m)$ and the base gates at $outputs(v)$. The disadvantage is that the position of the candidate gate is independent of the positions of gates directly connected to it and hence the wire cost associated with this dynamic updating is pessimistic.

The advantage of the second approach is that m will be placed at a position which causes minimum increase in the wire length with respect to its fanin and fanout which is a desirable feature. (This option corresponds to a constructive placement procedure for the network being mapped. Note that because of dynamic programming formulation, we are generating and storing as many constructive placement solutions as there are mapping solutions. A mapping solution along with its associated placement solution are determined after each logic cone is processed.) The disadvantages are that the *placePositions* for the as yet unmapped $outputs(v)$ do not have much correlation with the *mapPositions* of the gates actually showing up at the outputs of v in the final network and that the placement may become unbalanced (i.e., one with overlaps and locally 'congested' areas along with 'holes' in the layout plane). The latter problem can be reduced by repeating the global placement on the partially mapped network after a cone or a predetermined number of cones are processed. In that case, we can assign *placePositions* to *eggs* and *hawks* based on the new placement result. The first problem is more difficult to overcome. One solution is to perform a preprocessing pass on the network during which we record separately for each node v all possible $outputs(v)$ by examining every possible match in the network which has v as an input. During this preprocessing phase, we place the matches at the center of mass of their merged nodes. Clearly, this technique leads to a slow-down of *Lily* since we now need to consider all different matches at the $outputs(v)$ before choosing m.

When using *CM-of-Fans* option, depending on the wire length metric adopted, the problem can be solved efficiently or can become difficult. Consider Figure 3.2, which shows the enclosing rectangles for the fanin and fanout nets of match m at v. Given a norm and the coordinates of these fanin and fanout rectangles r, the problem is to find a point p which results in the minimum sum of distances between that point and the rectangles. In case of the Manhattan norm, the solution easily follows by observing that the distance function has a separable form with respect to the variables x and y. E.g., the x distance of point p from rectangle r can be written as:

$$f(x) = \frac{1}{2}(|r.ll.x - p.x| + |r.ur.x - p.x| - |r.ur.x - r.ll.x|)$$

where *ll* and *ur* refer to the lower left and the upper right of rectangle r. The constant term is dropped and the problem can be restated as: Find the point x such that $\sum_i |x_i - x|$ is minimum where x_i corresponds to either the left or the right corner point coordinates of each of the rectangles. The problem is a special case of solving for the median of a graph which is presented in [1]. It can be shown

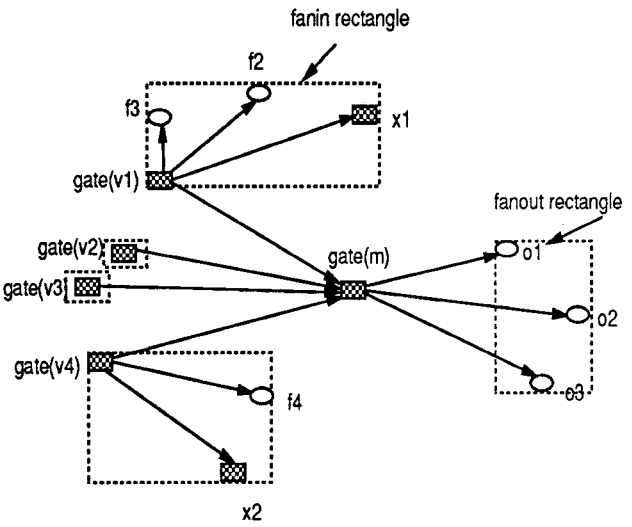

Figure 3.2: Dynamic updating of placement positions

that this problem, treating only a linear tree rather than a general graph, is very easy to solve; the solution is the median point for the sorted list of x_i's.

For the Euclidean norm, N rectangles partition the plane into N^2 subregions. In each subregion, the above optimization can be formulated as a quadratic optimization problem with linear constraints which can be solved efficiently. The global solution is obtained by comparing the cost of the best solution in each subregion and picking the minimum cost solution. Pruning of regions can reduce the number of subregions that must be considered. However, this still takes far more time than we can afford during the mapping process. Hence, an approximate solution is pursued. In particular, we represent each fanin/fanout rectangle by its center point, then the optimal point location problem is solved by computing the center of mass of these center points. Note that when constructing the fanin/fanout rectangles, we exclude nodes of $merged(v, m)$ from the fanin/fanout nets.

3.3 Fanin and Fanout Rectangles

We explain how the fanin rectangles for match m at v (which is a node in cone K_i) are constructed. (See Figure 3.1.) The key procedure is *add-true-fanout-recursively* which accepts a stem node, say v_i, and a fanout branch, say f_j, and finds the 'true fanout'(s) for the stem node along that branch. Here, 'true fanout' refers to a fanout of v_i that would be present had the mapping process been terminated after cone K_{i-1} was processed. A 'true fanout' is a *hawk*, a *nestling* or an *egg* which has v_i as its fanin. Due to logic duplication, it is possible to find more than one 'true fanout' along a given branch. For *hawks*, we use their *mapPositions*. For other nodes, we use their *placePositions*. (For example, the list of 'true fanout's of node v_1 consists of nodes u_1, x_1, f_2 and f_3.)

We add v_i to the list of nodes and delete those 'true fanout's of v_i which are covered by the current match m. (The new list for v_1 consists of x_1, f_2, f_3 and v_1.) Next, we build a minimum rectangle enclosing all nodes in the new list. Note that we use *mapPositions* of v_i and its *hawk* fanouts and *placePositions* of all other nodes in the list. Construction of the fanout rectangle is easy since outputs of $gate(m)$ are *eggs* due to depth first search ordering of processed nodes. We directly use their *placePositions* to build the fanout rectangles.

3.4 Wire Cost Estimation

After positioning $gate(m)$, we must estimate the wire cost associated with the matching of m at node v. We have implemented two options. For each fanin v_i, we include $gate(m)$ in the fanin rectangle for v_i and calculate the half perimeter length of the fanin rectangle divided by 'true fanout' count at v_i (in order to avoid duplicate accounting for the wire cost) to get the expected wire length contributed by input net to v_i. This length is then multiplied by the ratio of minimum rectilinear Steiner tree length to half perimeter of enclosing rectangle as given by [3]. We have also implemented another wiring model based on finding the rectilinear spanning tree connecting all "pins" on a given net.

3.5 Cone Ordering

The 'true fanout's corresponding to the *hawks* will necessarily exist in the final network. Other 'true fanout's are tentative, in the sense that they may not exist in the final network. However, we use all 'true fanouts' for constructing the fanin rectangles as described above. Therefore, we should come up with an ordering of output cones that minimizes number of references to the 'true fanout's which have not been mapped yet. Reconvergent stem nodes whose reconvergence region is a subset of exactly one logic cone give rise to *egg* or *nestling* 'true fanout's inside the current logic cone. However, we cannot avoid this situation. Therefore, we set out to find an ordering that minimizes the number of references to the *eggs* outside the current logic cone. This problem may be restated as follows: Find an output cone ordering such that the sum over all cones of the number of exit lines from any cone to all unmapped cones is minimized.

More formally, let $(\pi_1, \pi_2, \cdots, \pi_n)$ denote a linear ordering on cones $K_1, K_2, \cdots, K_n$. Then, it is the desired ordering if it minimizes the following sum:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(K_{\pi_i}, K_{\pi_j})$$

where $E(K_{\pi_i}, K_{\pi_j})$ denotes the number of exit lines from K_{π_i} to K_{π_j}. We build an $n \times n$ matrix, M, where we store $E(K_i, K_j)$ in its ij position. Note that M is a symmetric matrix with all diagonal entries equal to zero. The desired ordering is obtained by recursive application of the following operations: Find a row, i, with minimum row sum $\sum_{j=1}^{n} E(K_i, K_j)$; push its corresponding primary output cone into a queue; delete row i and column i from M. This procedure will find the optimum linear ordering of output cones for the specified objective function.

4 Technology Mapping for Minimum Delay

In the delay mode, the best mapping at a node is determined based on the arrival time of the signal at the node output. As technology scales down, the contribution of wiring to the delay becomes significant, and even dominating [4, 13]. Hence, it is only natural that we have attempted incorporating wiring delay into the calculation of the arrival time.

4.1 Arrival time calculation

Consider a gate g with output line y and input lines i, $i = 1 \cdots p$. Let g fanout to inputs of g_j. In a simple linear delay model, the delay through g is a linear function of its output load capacitance C_L. The slope of this linearity can be thought of as the *output resistance* and the offset (at zero C_L) can be thought of as the *intrinsic delay* through g. In general, the delays from different inputs to the output are different. We represent the intrinsic delay from input i to y by I_i and the output resistance at y corresponding to input i by R_i. Note that I_i and R_i have separate values each for rising and falling delays.

Based on this model, the arrival time [7] at y from input i, t_{y_i}, can be easily calculated as $t_{y_i} = t_i + I_i + R_i C_L$ where t_i is the arrival time at input line i. (Again, note that arrival times have to be calculated separately for rising and falling delays). Using a worst case analysis, the *output arrival time* at y, t_y is defined as the the time at which all signals from input lines i will be available at y and is given by $t_y = max\{t_{y_i}\}$ computed over all $i, i = 1 \cdots p$. Combining the above two equations, we have the recursive formula for the output arrival time as $t_y = max\{t_i + I_i + R_i C_L\}$ computed over all $i, i = 1 \cdots p$. This calculation for the arrival time requires that the value of C_L be known.

4.2 Output load capacitance

C_L is the equivalent capacitive load at y. This capacitance is modeled as $C_L = \sum_{j=1}^{n} C_j + C_w$ where C_j denotes the capacitance at the input of fanout gate g_j, and n is the number of fanout nodes. C_w represents the capacitance due to the interconnections which connect g to its fanout nodes. The wiring resistance is very small and is therefore ignored.

Let q be the input of the fanout gate g_j to which y is connected. Since we have modeled the interconnections by a lumped capacitance, $t_y = t_q$; i.e., we have assumed that output arrival time at y and the input arrival time at q are identical.

In MIS, C_w is modeled as a function of the n. (A simple function would be linear in n, with a user specified proportionality constant.) In *Lily*, we estimate C_w based on the wiring information and C_w is modeled as a lumped capacitance proportional to the estimated output net length. If X and Y are the horizontal and vertical interconnection lengths for the nets, the capacitance is calculated as $C_h X + C_v Y$, where C_h and C_v are the capacitance per unit length of the horizontal and vertical interconnects respectively. X and Y can be determined using models described in Section 3.3.

4.3 Updating the arrival time

During the mapping process, when we match m at node v, the fanouts of v are not yet mapped. This implies that the load C_L, at the output of *gate(m)* cannot be determined exactly. This problem can be handled by assuming a *constant load*; i.e., all types of gates are assumed to have the same input parasitic capacitance. This assumption is also adopted in MIS2.1. (Most gates in the 3μ MSU standard cell library have an input capacitance of 0.25 pF [12]). However, in order to calculate the wiring capacitance, we need to know the *position* in addition to the type of gate at the fanout. This is not possible and we instead use the nodes in the $N_{inchoate}$ as the fanouts. This procedure gives rise to inaccuracies in the arrival time calculation.

To prevent this inaccuracy from propagating through, we make the following observation: The capacitance at the output of *inputs(v,m)* is now known because we know the type and position of their fanout gate which is *gate(m)*, the current match. If we update the output arrival times of *inputs(v,m)*, then the input arrival time of *gate(m)* is accurate. The splitting of the arrival time calculation into load dependent and load dependent parts makes such an update easy. This can be thought of as splitting the gate g into p load independent parts LI_i and one load dependent part LD. Each input i has an associated LI_i. The LI's have zero output resistance and LD has zero intrinsic delay. Corresponding to each input i, we define the *block arrival* time at g as $b_i = t_i + I_i$. The output arrival

time can now be defined in terms of the block arrival times and is given by $t_y = max\{b_i + R_i C_L\}$. The advantage of this splitting is that only the $R_i C_L$ part has to be redone for different loads - b_i's remain the same.

4.4 Mapping for minimum delay

Consider the mapping at node v in Figure 3.1. We have already calculated the block arrival times at v_i. The mapping proceeds in the following manner:

1. For each $v_i \in inputs(v, m)$, the output arrival time at $gate(v_i)$ is recalculated. This computation uses the block arrival times at v_i and the *current* load at the output of v_i. We find the list of 'true fanout' nodes for v_i and add match m to the list. The current load, seen at v_i, is calculated from this list. We use the input capacitance of $gate(m)$ for calculating C_L and its *mapPosition* for calculating C_w. For a *hawk* in the list, we use the input capacitance and the *mapPosition* for $gate(hawk)$. For an *egg* or *nestling* in the list, we use the input capacitance and the *placePosition* for its base function gate.

2. The block arrival times at $gate(m)$ and corresponding to each input v_i are computed.

3. Using the base function gates at the fanout of v, the output capacitance load of $gate(m)$ is calculated.

4. The output arrival time at $gate(m)$ is calculated using the block arrival time and the output load.

5. The output arrival time at $gate(m)$ is compared with the output arrival time of other possible matches at v. The matching with the lowest output arrival time is chosen. The match and its block arrival times are stored at v.

5 Experimental Results and Discussions

Consider using the traditional mapping schemes on a given design but with two different target libraries. Both libraries implement the same functions. However, the 'tiny' library has gates up to 3 inputs while the 'big' library has gates up to 6 inputs. Clearly, mapping with 'tiny' library contains many more gates and nets. Its active cell area and total chip area are, in general, larger. The 'big' library has much smaller active cell area, but its routing complexity is high. Consequently, the final chip area after placement and routing can be as large as that obtained using the 'tiny' library. Let A_{tiny} and A_{big} denote the chip area obtained by traditional mappers using 'tiny' or 'big' libraries. Similarly, let W_{tiny} and W_{big} denote the total interconnection length. Now, if we use a mapping technique such as presented in this paper along with the 'big' library, we will find a mapping solution with number of gates in between those of 'tiny' and 'big' libraries but with $\widehat{A} < min(A_{tiny}, A_{big})$ and $\widehat{W} < min(W_{tiny}, W_{big})$.

We wanted to show that by integrating technology mapping and gate placement, one can improve the quality of mapping both in terms of layout area and circuit performance. In order to provide a fair basis for comparison, we went through two pipelines to produce results: 1) Read in the optimized circuit, run MIS technology mapper in area and timing mode, write mapped circuit to the database, assign locations to I/O pads, do detailed placement and routing. 2) Read in the optimized circuit, assign locations to I/O pads, run *Lily* in area and timing mode, write mapped circuit to the database, do detailed placement and routing. In both cases we use the same placement, pin assignment and routing tools. Note that the first option which is the standard MIS pipeline cannot make use of the location of pads during the technology mapping process.

Ex.	MIS2.1			Lily		
	inst. mm^2	chip mm^2	wire mm	inst. mm^2	chip mm^2	wire mm
9symml	0.27	0.68	76.1	0.29	0.69	74.1
C1908	0.69	2.23	245.6	0.71	2.15	233.5
C3540	1.74	6.90	874.1	1.85	6.52	813.7
C432	0.34	0.95	105.7	0.37	0.93	101.2
C499	0.66	1.85	192.3	0.65	1.88	189.3
C5315	2.16	10.92	1303.0	2.20	10.48	1275.7
C880	0.62	2.11	236.8	0.64	1.81	193.9
apex6	0.98	3.99	474.7	1.00	3.81	454.5
apex7	0.34	0.94	107.7	0.35	0.88	96.1
b9	0.19	0.41	38.9	0.20	0.38	36.8
apex3	2.17	20.86	2880.3	2.24	19.90	2685.2
duke2	0.67	2.47	293.7	0.69	2.31	265.3
e64	0.41	1.24	137.4	0.41	1.13	129.7
misex1	0.083	0.16	15.5	0.088	16.9	16.1
misex3	0.87	3.68	484.9	0.86	3.37	424.4

Table 1: Comparison of the total instance area, final chip area and interconnection length after detailed routing between MIS2.1 and Lily.

Ex.	MIS2.1		*Lily*	
	inst. mm^2	delay ns	inst. mm^2	delay ns
9symml	0.44	10.23	0.42	9.27
C1908	1.48	26.04	1.57	24.26
C432	0.59	24.11	0.62	21.59
C499	1.20	15.79	1.38	17.27
C5315	4.61	29.81	4.79	25.98
C880	0.98	21.46	1.12	18.63
apex7	0.48	8.85	0.57	8.64
b9	0.29	5.04	0.28	4.74
duke2	1.15	22.29	1.16	17.81
e64	0.56	22.80	0.71	20.37
misex1	0.14	7.30	0.15	6.72
misex3	1.43	22.21	1.67	19.56

Table 2: Comparison of the total instance area and longest path delay results between MIS2.1 and Lily (1 μ technology).

Table 1 depicts comparisons between our results and those of MIS2.1 in terms of active cell area, total chip area and total interconnection length (in area mode). In general, our mapper tends to use smaller gates, larger active cell area (avg. 1%) but smaller total chip area (avg. 5%) and interconnection length (avg. 7%) due to reduced routing complexity.

Table 2 shows our delay optimized mapping results and those obtained using MIS2.1 (in timing mode). The delays are in arbitrary units, and are based on a 1μ standard cell library. Since information on a real 1μ library was not available, we scaled the delay, gate capacitance and wiring capacitance of 3μ technology [12]. Both MIS2.1 and *Lily* delays are computed after detailed placement, and the wiring delays are included during the delay calculation. *Lily* shows an average delay improvement of 8% compared to MIS2.1.

Note that our dynamic wire length estimation procedure is not always accurate (as seen by poor results for misex1 in Table 1 and C499 in Table 2). This indicates that our procedure does not capture

the full effects of layout during synthesis. In such cases, we could repeat the mapping with reduced wire cost weight to obtain better solutions.

We used GORDIAN [21] package for global placement, *CM-of-Fans* option for dynamic placement update, half perimeter length of the net enclosing rectangle for wire length estimation, pad placement program described in [20], TimberWolf 4.2 [6] global router and YACR [5] detailed router. The placement package generates a global placement for the pre-mapped inchoate network, of C5315, with 1892 gates in about 3 minutes on a DEC3100. The *Lily* run time - including premapping, pad placement, global placement of the inchoate network, mapping, detailed placement of the mapped circuit with 713 gates for this example is about 10 minutes.

We have observed that *Lily* yields better mapping solutions (e.g., compared to MIS2.1 mapper) when the routing complexity for the logic circuit is high and the target library contains large gates (number of fanin nodes > 4). In addition, the initial pad placement - prior to technology mapping - influences the degree of wire length reduction that is achievable by *Lily*.

Currently, *Lily* does not perform fanout optimization. In addition, its delay model is a load independent delay model which tries to overcome some of the shortcomings of a load independent delay calculation by using information about the mapped portion of the inchoate network. As in MIS2.2 [17], we could perform a preprocessing pass during which we record for each node all possible load values at that node by examining every possible match or perform a postprocessing pass to derive fanout trees.

We believe that layout driven technology mapping (and in general, layout driven logic synthesis) is a promising direction for research in coming years and that there are still many issues which must be addressed. We hope to improve and extend our model. A logical extension would be to consider layout effects during kernel extraction and node decomposition. Work presented in [18] seems relevant.

6 Conclusions

In this paper we have described *Lily*, a technology mapper implemented on top of MIS2.1, which integrates *layout* area and delay into the technology dependent phase of logic synthesis. We have shown techniques and ideas to incorporate the estimated wiring information into the dynamic programming algorithm used for the tree matching. The key idea in our work is to do a fast global placement of the logic network in its inchoate form. This initial global placement guides the wiring estimation. As the mapping proceeds along, the initial placement is dynamically updated, so that nodes in the network which are processed later, can use more accurate information. Other techniques such as optimal pad placement and ordering of the logic cones also help in arriving at a mapping solution which has better performance after placement and routing. Both delay and area optimization techniques have been implemented and the preliminary results are encouraging. Our work is a first step towards fusing layout considerations into logic synthesis, and we hope to build on this work and extend it to consider layout issues during the technology independent optimization phase as well.

7 Acknowledgements

We deeply appreciate the valuable discussions with Professor M. Marek-Sadowska of University of California, Santa Barbara and Dr. Ulrich Lauther of Siemens Corporate Research Division and thank our research advisor, Professor Ernest S. Kuh, for his continuous advice, support and encouragement throughout the course of this project. We thank Professor Kurt Antreich of Technical University of Munich for providing us with the GORDIAN placement package, Stefan Mayhofer for putting the package in our system, Kamal Choudhary for helpful discussions, Hamid Savoj for providing us with optimized circuits and George Carvalho for helping with the implementation. This research was sponsored by the National Science Foundation, under Grant No. MIP 88-03711 and by the Semiconductor Research Corporation, under Grant No. 91-DC-008.

References

[1] S. L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," *Oper. Res.*, 12, pp. 450-459, 1964.

[2] A. Aho and S. Johnson, "Optimal code generation for expression trees," *J. ACM*, pp. 488-501, July 1976.

[3] F. R. K. Chung and F. K. Hwang, "The largest minimal rectilinear Steiner trees for a set of n points enclosed in a rectangle with given perimeter," *Networks*, vol 9, pp. 19-36, 1979.

[4] K. C. Saraswat and F. Mohammadi, "Effect of scaling of interconnections on the time delay of VLSI circuits," *IEEE Trans. on Electron Devices*, vol ED-29, pp. 645-650, 1982.

[5] Jim Reed, "YACR: Yet Another Channel Router," Master's Report, University of California, Berkeley, February 1985.

[6] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE J. of Solid State Circuits*, vol 20, no. 2, pp. 510-522, April 1985.

[7] M. Burstein and M. N. Youssef, "Timing influenced layout design," *Proc. 22-nd Design Automation Conference*, pp. 124-130, 1985.

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

[9] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang, "Technology mapping in MIS," *Proc. Int. Conf. CAD (ICCAD-87)*, pp. 116-119, Nov. 1987.

[10] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: Multiple-level interactive logic optimization system," *IEEE Trans. on CAD*, vol 6, no. 6, pp. 1062-1081, Nov. 1987.

[11] G. Hachtel, M. Lightner, R. Jacoby, C. Morrison, P. Moceyunas, and D. Bostick, "Bold: The boulder optimal logic design system," in *Hawaii Int. Symp. on Systems Sciences*, 1988.

[12] D.V. Heinbuch ed., *CMOS 3 Cell Library*, Addison-Wesley Publishing Company, 1988.

[13] Y. A. El-Mansy and W. M. Siu, "MOS technology advances," in *Handbook of Advanced Semiconductor Technology and Computer Systems*, G. Rabbat ed., Van Nostrand Reinhold Company, pp. 229-259, 1988.

[14] R. S. Tsay, E. S. Kuh, and C.P. Hsu, "PROUD: A sea-of-gates placement algorithm," *IEEE Design and Test of Computers*, pp. 318-323, Dec. 1988.

[15] M. Pedram and B. T. Preas, "Interconnection length estimation for optimized standard cell layouts," *Proc. Int. Conf. CAD (ICCAD-89)*, pp. 390-393, 1989.

[16] 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.

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

[18] P. Abouzeid, K. Sakouti, G. Saucier and F. Poirot, "Multilevel synthesis minimizing the routing factor," *Proc. 27-th Design Automation Conference*, pp. 365-368, 1990.

[19] M. Pedram, M. Marek-Sadowska and E. S. Kuh, "Floorplanning with pin assignment," *Proc. Int. Conf. CAD (ICCAD-90)*, pp. 98-101, 1990.

[20] M. Pedram, N. Bhat and K. Choudhary, "LILY: A layout-driven approach to technology mapping," *UCB ERL Memo, Electronics Research Laboratory, Univeristy of California, Berkeley.*, M90/97, 1990.

[21] J. M. Kleinhans, G. Sigl, F. M. Johannes and K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. on CAD*, vol 10, no. 3, pp. 356-365, March 1991.