LAYOUT-ORIENTED TECHNOLOGY MAPPING AND IO PAD ASSIGNMENT

by

Massoud Pedram, Narasimha Bhat, and Kamal Chaudhary

Memorandum No. UCB/ERL M90/97

14 November 1990

LAYOUT-ORIENTED TECHNOLOGY MAPPING AND IO PAD ASSIGNMENT

by

Massoud Pedram, Narasimha Bhat, and Kamal Chaudhary

Memorandum No. UCB/ERL M90/97

14 November 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

LAYOUT-ORIENTED TECHNOLOGY MAPPING AND IO PAD ASSIGNMENT

by

Massoud Pedram, Narasimha Bhat, and Kamal Chaudhary

Memorandum No. UCB/ERL M90/97

14 November 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Contents

Table of Contents i

List of Figures ii

1 Layout-oriented Technology Mapping 1
- 1.1 Introduction and Motivation . . . 1
- 1.2 Terminology . . . 2
- 1.3 Algorithm Overview . . . 3
- 1.4 Technology Mapping for Minimum Layout Area . . . 6
  - 1.4.1 Global Placement . . . 6
  - 1.4.2 Incremental Updating of Placement . . . 8
  - 1.4.3 Fanin and Fanout Rectangles . . . 10
  - 1.4.4 Wire Cost Estimation . . . 10
  - 1.4.5 Cone Ordering . . . 11
- 1.5 Technology Mapping for Minimum Delay . . . 11
  - 1.5.1 Arrival time calculation . . . 11
  - 1.5.2 Output load capacitance . . . 12
  - 1.5.3 Updating the arrival time . . . 12
  - 1.5.4 Estimation of wiring capacitance . . . 13
  - 1.5.5 Mapping for minimum delay . . . 13
  - 1.5.6 Inaccuracies in the delay model . . . 13
- 1.6 Experimental Results and Discussions . . . 14
- 1.7 Conclusions . . . 15

2 IO Pad Assignment based on the Circuit Structure 17
- 2.1 Introduction . . . 17
- 2.2 Solution Technique Overview . . . 18
- 2.3 Experimental Results . . . 20
- 2.4 Conclusions . . . 20

Bibliography 21

List of Figures

1.1 4
1.2 5
1.3 6
1.4 7
1.5 9

Chapter 1

Layout-oriented Technology Mapping

Massoud Pedram, Narasimha Bhat

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 tools 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 an incrementally updated global placement of the Boolean network. The incremental 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.

1.1 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 tools currently available split this task into two phases – a technology independent phase and a technology dependent phase [5]. Within each phase, some measure of the cell area and cell delay are used as objectives for minimization. It is assumed that wiring optimization can be handled efficiently in the physical design phase. However, decisions made during the logic synthesis phase may limit the optimization potential of physical design tools. For example, excessive factorization based on common factor extraction during the technology independent phase of logic synthesis can lead to gates with high fanout and large interconnection lengths for logic implemented by standard cells. 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 the path delay. Ignoring propagation delay through wires has introduced inaccuracy in the timing analysis performed during the technology mapping.

With recent studies [4] 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. This problem can be stated as follows. Given a Boolean network representing a combinational logic circuit optimized by the 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 systems like DAGON [6] 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[8]. DAGON and MIS technology mapper generate circuits with small active cell area but ignore area and delay contributed by the interconnections between the gates.

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 the *layout* area and delay are minimized. Layout area is the sum of gate areas and routing area. 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 an incrementally updated global placement of the Boolean network. The incremental update does not restrict the dynamic programming approach adopted in technology mappers such as DAGON and MIS.

The rest of the paper is organized as follows. In Sections 2 and 3, we state the terminology and notations used throughout the paper and give an overview of the algorithm. In Sections 4 and 5 we discuss technology mapping targeted toward area minimization and delay minimization, respectively. Experimental results and concluding remarks are presented in Sections 6 and 7.

1.2 Terminology

The approach of DAG covering for technology mapping can be summarized as follows. A set of base functions is chosen such as a two-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 NOT 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 the tree boundaries and, as a result, may duplicate logic. The MIS technology mapper implements DAGON as a subset.

Consider Figure 1 which shows an example $N_{inchoate}$ at some point during the mapping process. Assume that we have processed cone K_1 corresponding to the 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. Ongoing to the nature of dynamic programming, 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 bird's life cycle as an *egg* and later become a *hawk*. (See Figure 2.) At the end of mapping procedure, only *hawk*s and *dove*s remain. This classification will be used later to describe the construction of fanin rectangles which are needed for updating the placement positions and estimating the wire lengths.

At the beginning of mapping process, every node is an *egg* node. We assign to these nodes initial *placePosition*s based on a placement of $N_{inchoate}$. As the mapping process advances, we calculate and assign new *mapPosition*s to the mapped nodes. In section 4, we will describe two methods for calculating these *mapPosition*s.

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 the output line of a node in K_i and and input line of a node which is not in K_i.

1.3 Algorithm Overview

We justify incorporating the wiring estimates into technology mapping for area optimization by pointing out the problems associated with minimizing only the active cell area. A gate with high fanin count which has minimum area and covers a sub-network in $N_{inchoate}$ can result in wiring congestions during the detailed routing phase since many wires (as many as there are fanin and fanout nodes for the gate) are forced to converge to the same gate boundary in the layout plane. In addition, such a matching may result in excessive wire length depending on how the fanin gates are connected to other gates in the circuit.

Figure 3 shows a small portion of $N_{inchoate}$. Our task is to transfer the signals from source nodes s_i to the sink node t using minimum wire length. For the sake of discussion, assume that positions of s_i and t are pre-determined. The decision problem is as follows. "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

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

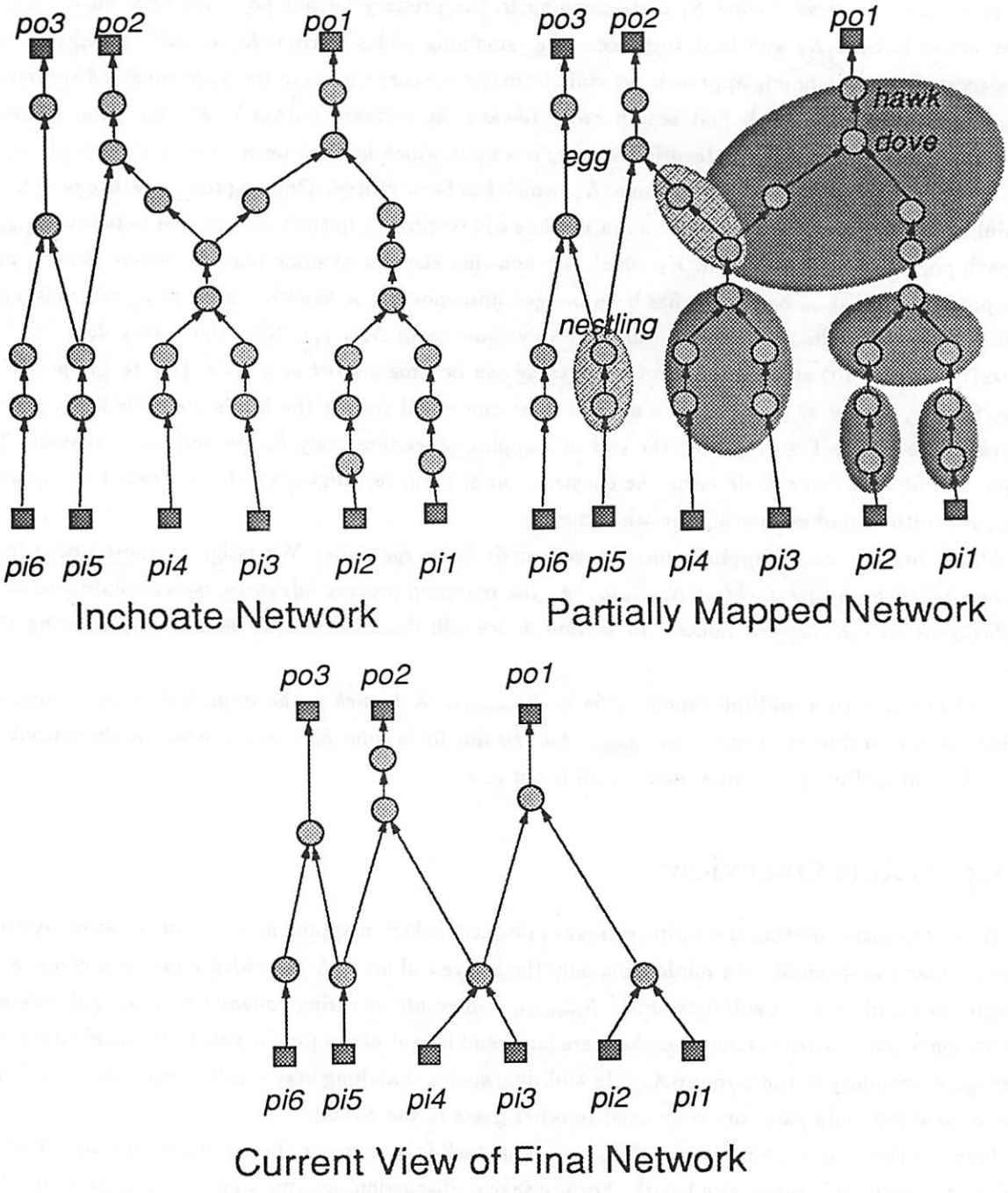

po3
po2
po1
pi6
pi5
pi4
pi3
pi2
pi1
Inchoate Network
egg
hawk
dove
nestling
Partially Mapped Network
Current View of Final Network

Figure 1.1:

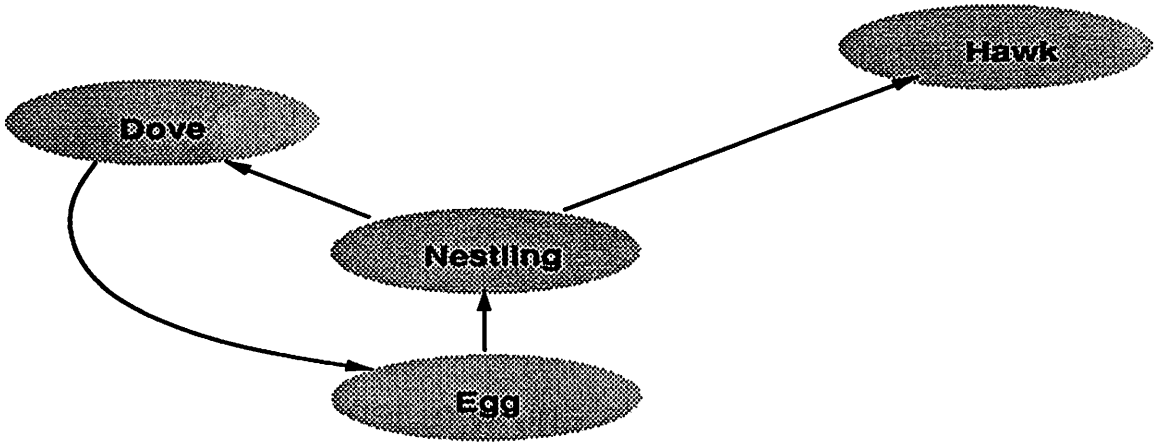


Figure 1.2:

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 3. 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.

Lily integrates I/O pad placement, technology mapping, gate placement and interconnection length estimation processes in a systematic and consistent manner. The key component of and the starting point for *Lily* is a global placement obtained for $N_{inchoate}$ which captures the structure of the network on the plane. This global placement solution contains overlapping gates and has not been adapted to a specific design style, that is, gates are not forced to fall into rows or slots. However, it is a *balanced* placement where gates are uniformly distributed over the convex hull defined by the external I/O pads for the circuit. *Lily* reads positions for the unmapped nodes in $N_{inchoate}$ from the global placement solution and uses this information to estimate wiring cost during the mapping process. As the mapping process advances, the number of nodes and the structure of $N_{inchoate}$ gradually changes and *Lily* updates its view of the Boolean network and its corresponding global placement on layout plane. We will present techniques to incrementally update the Boolean network and initial global placement and demonstrate that there is a good correlation between the the incrementally updated placement solution and a global placement obtained directly for N_{mapped}.

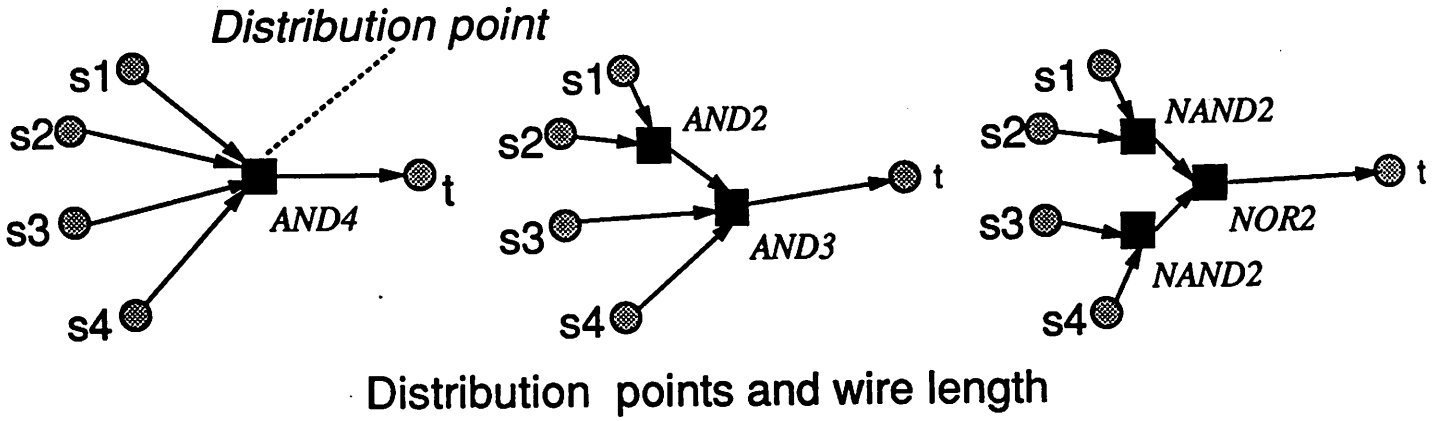


Figure 1.3:

1.4 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 the 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 4.) This cost consists of two components as follows.

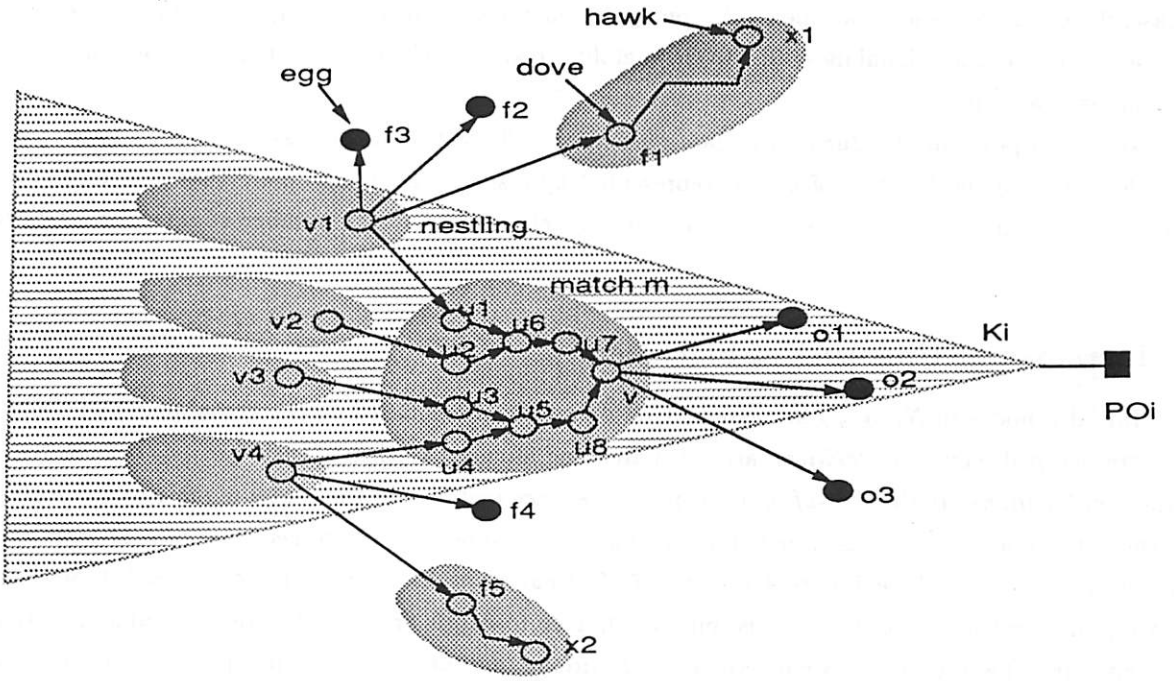
$$areaCost(v,m) = area(gate(m)) + \sum_{v_i \in inputs(v,m)} areaCost(v_i)$$

$$wireCost(v,m) = wire(gate(m), gate(v_i)) + \sum_{v_i \in inputs(v,m)} wireCost(v_i)$$

Here, $inputs(v,m)$ refers to the list of nodes of G which correspond to the inputs of m. $gate(m)$ is the physical gate corresponding to the 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 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 $gate(v_i)$.

1.4.1 Global Placement

We use a global placement procedure [10, 11] 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 [14]. 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 chip-level considerations (as in [15]) or by a bottom-up I/O pin assignment procedure driven by structure of $N_{inchoate}$ (as described in Chapter 2).



View of the Inchoate Network during Mapping

Figure 1.4:

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 a combined optimization and bi-partitioning technique 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 subregion.

Such a global placement is desirable for two reasons. Firstly, the incentive for the global placement is to capture the structure of the Boolean network on 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 incremental 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 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.

1.4.2 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* option, 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 4.) 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 from the distances between $gate(m)$ and the best function gates matching at $inputs(v, m)$ 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 incremental 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 in 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

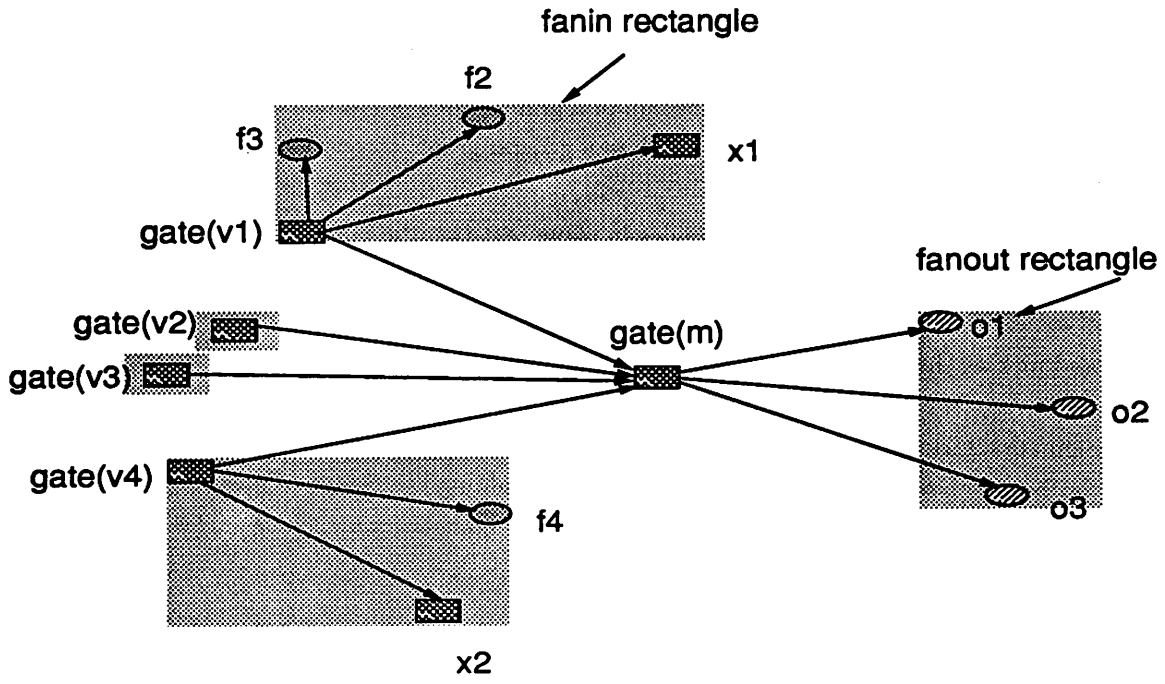


Figure 1.5:

outputs(v) may not have much correlation with the *mapPosition* of the gates actually showing up at the outputs of v in the final network and that the incremental 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 *egg* nodes and *hawk* nodes 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 *output*(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. (See [9].)

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 5, 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 and that the distance of point p from rectangle r can be written as:

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

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 [12]. It can be shown 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.

In the case of Euclidean norm, N rectangles in plane 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 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 center points for the rectangles. Note that when constructing the fanin/fanout rectangles, we exclude nodes of *merged*(v, m) from the fanin/fanout nets.

1.4.3 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 4.) 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* node which has v_i as its fanin, a *nestling* node or an *egg* node. Due to logic duplication, it is possible to find more than one 'true fanout' along a given branch. For *hawk* nodes, 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 fanout rectangle is easy since outputs of *gate*(m) are *egg* nodes due to depth first search ordering of processed nodes. We directly use their *placePositions* to build the fanout rectangles.

1.4.4 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 [13]. We have also implemented another wiring model based on finding the rectilinear spanning tree connecting all "pins" on a given net.

1.4.5 Cone Ordering

The 'true fanout's corresponding to the *hawk* nodes 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 gives 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 *egg* nodes 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 the term in double sum 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.

1.5 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 [2, 4]. Hence, it is only natural that we have attempted incorporating wiring delay into the calculation of the arrival time.

1.5.1 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 [3] 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.

1.5.2 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, thereof, 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, $t_y = t_q$, that is, 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. (See section 5.4.)

1.5.3 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 is also adopted in MIS version 2.1. (Most gates in the 3μ MSU standard cell library have an input capacitance of 0.25 pF [1]). 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 results in an inaccurate 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 LIs 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_is remain the same.

1.5.4 Estimation of wiring capacitance

A net is modeled as a lumped capacitance (and the resistance of the routing is ignored). 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. The wiring capacitance is thus determined by estimating the horizontal and vertical extents of the net. This can be determined using models described in Section 4.4.

1.5.5 Mapping for minimum delay

Consider the mapping at node v in Figure 4. 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 C_L calculation and its *mapPosition* for C_w calculation. For a *hawk* node in the list, we use the input capacitance and the *mapPosition* for *gate*(*hawk*). For an *egg* or *nestling* node in the list, we use the constant 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.

1.5.6 Inaccuracies in the delay model

An accurate arrival time calculation for a gate requires that the positions of the gate and its fanout nodes are known. Given the positions, the routing length can be estimated to a fair degree of accuracy. However, it is not possible to determine the fanout positions in the dynamic programming algorithm. By assuming the

fanouts to be base function gates in the inchoate mapping, and by using the inchoate placement positions, inaccuracy is introduced. To reduce this inaccuracy, the arrival time calculation is repeated at $gate(v_i)$. This second calculation is more accurate than the one which was calculated when v_i itself was being mapped.

1.6 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 minimized Boolean network, 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 minimized Boolean network, 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 pipeline cannot make use of the location of pads during the technology mapping process. Table 1 depicts comparisons between our results and those of MIS2.1 in terms of number of gates, active cell area, total chip area and total interconnection length. In general, our mapper tends to use smaller gates, larger active cell area but smaller total chip area (3-5%) and interconnection length (6-10%) due to reduced routing complexity).

Table 2 shows our delay optimized mapping results and those obtained using MIS. 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 [1]. Both MIS2.1 and *Lily* delays are computed after detailed placement, and the wiring delays are included during the delay calculation.

We used GORDIAN [10] package for global placement, *CM-of-Fans* option for incremental placement update, minimum rectilinear spanning tree for wire length estimation and pad placement program described in next Chapter. These options produce best results. The *Gordian* placement package generates a global placement for a pre-mapped inchoate network with 1915 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 198 gates for this example is about 7 minutes.

We have observed that *Lily* yields better mapping solutions (e.g., compared to MIS2.1 mapper) when

Example	MIS2.1				Lily			
	num inst.	total inst. area(mm^2)	total chip area(mm^2)	wire length(mm)	num inst.	total inst. area(mm^2)	total chip area(mm^2)	wire length(mm)
9symml	73	0.215	0.431	50.459	74	0.215	0.407	48.576
C1355	206	0.582	1.159	134.189	198	0.568	1.069	127.490
C1908	245	0.659	1.313	168.658	259	0.682	1.284	162.331
C5315	689	2.001	3.976	763.490	712	2.038	3.811	732.460
C7552	1140	3.063	6.081	946.810	1215	3.057	5.887	892.112
C880	189	0.529	1.055	140.763	197	0.542	1.012	133.774
alu4	82	0.213	0.427	39.953	85	0.218	0.413	38.823
apex6	305	0.877	1.746	363.5	354	0.959	1.681	298.417
apex7	123	0.318	0.636	77.407	146	0.353	0.624	70.796
duke2	185	0.512	1.022	155.611	191	0.522	0.985	151.443
o64	20	0.112	0.225	28.654	32	0.132	0.219	26.540
rot	372	0.953	1.897	332.960	382	0.971	1.812	312.417

Table 1.1: Comparison of of wiring results MIS2.1 versus Lily.

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, 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. The main thrust of this work is to show that by including the effect of interconnects during mapping, we can synthesize circuits with better performance and area. Although we have implemented *Lily* at the level of MIS2.1, the ideas described throughout the paper can be applied to *MIS*2.2 as well.

We believe that layout driven technology mapping (and in general, layout driven logic synthesis) is a promising direction for research in the incoming years and that there are still many issues which must be addressed. We hope to improve and extend our model. Work presented in [16] seems relevant. We hope to improve our preliminary results by the time of final submission.

1.7 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,

Example	MIS2.1		*Lily*	
	delay	area	delay	area
9symml	142.56	4416	135.16	4304
C1908	404.25	16512	374.74	14456
C5315	398.53	49768	358.22	37504
C880	273.58	12248	237.89	10456
apex2	136.25	6160	124.75	6272
apex6	384.88	16928	378.01	17768
apex7	133.38	5112	114.13	4984
b9	79.04	3296	65.39	3080
sao2	131.23	3160	120.76	2992

Table 1.2: Comparison of timing results for MIS2.1 versus Lily.

the initial placement is incrementally 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. As mentioned earlier, our work is a first step towards including interconnection considerations into the logic synthesis phase, and we hope to build on this work and modify it to be more robust and effective.

Acknowledgements The authors would like to acknowledge the support received for this research from SRC Grant 90-DC-008.

Chapter 2

IO Pad Assignment based on the Circuit Structure

Massoud Pedram, Kamal Chaudhary

Abstract

We present an IO pad assignment technique for placing off-chip IO pads. This technique makes use of the information available in the circuit structure. It can be run prior to technology mapping or placement procedures. Extension to timing-driven IO-pad assignment is straight-forward. The preliminary results are encouraging.

2.1 Introduction

A common procedure for generating layout from the register transfer level description of an electronic circuit is as follows. Circuit is partitioned into a number of combinational blocks. Each block is simplified using technology-independent transformations so that an estimate of block area is minimized and the required arrival time constraints at the primary outputs are satisfied. Next, each block is mapped into a target library. The resulting net list is, then, passed to detailed placement and routing procedures which generate the layout.

During the technology mapping phase, knowledge of off-chip IO pad locations can be used to calculate the interconnection cost as discussed in Chapter 1. The standard technology mapping schemes ignore the effect of wiring during the graph-covering procedure and only minimize the active gate area. We have already shown the importance of including the wiring cost into the mapping process. Therefore, it is useful to have an IO pad assignment procedure which is driven by the circuit structure.

When solving the gate placement problem, fixed placement of off-chip IO pads is required prior to placing the gates. This is because in the absence of off-chip IO pads, the gates collapse to the center of chip. In addition, the quality of detailed placement result strongly depends on the initial off-chip IO pad placement. A common approach is to use an arbitrary (or random) pad placement prior to detailed placement and then

improve the pad locations based on the detailed routing result. The two phases are iterated until an acceptable placement solution is generated. In general, this two-phase approach is undesirable due to the following reasons. Even if convergence is achieved, the solution is heavily influenced by the initial pad placement. The iteration process is costly and time-consuming.

The IO pad placement is even more important if there are path delay constraints from primary inputs to primary outputs. In that case, the initial location of the IO pads will greatly influence the quality of timing driven placement obtained. In particular, a poor pad placement may result in infeasible placement solution.

In this chapter, we present a novel technique for deriving a good initial pad placement. This technique, which is based on the analysis of the circuit structure and path delay constraints, uses IO clustering and linear-sum assignment to assign locations to IO pads.

2.2 Solution Technique Overview

We assume that the circuit is specified in the form of a directed acyclic graph (DAG), that is, a *Boolean network* prior to technology mapping or a *directed net list* prior to detailed placement. For each primary output po_i, we traverse the circuit in a depth-first order and identify its primary input support (i.e., its transitive fanin cone, K_i. Next, we derive a linear ordering on the outputs as follows. Corresponding to each primary output po_i, we create a block B_i. Corresponding to each primary input pi_j in the support set of po_i, we create a pin p_j and attach it to block B_i. Consequently, we have a net list representing primary outputs and their transitive primary inputs. We generate a linear placement of blocks B_i which minimizes the total net span. This solution corresponds to a linear ordering on the primary outputs that maximizes 'proximity' among their support set.

We distribute the primary outputs over the chip boundary in the order derived above. The distance between consecutive outputs, po_i and po_j, is estimated as follows. We traverse the DAG in topological order and assign a *level* to each node. The level of a node is the maximum distance from any primary input to that node. Let K_i and K_j denote the transitive fanin cone of po_i and po_j. We find a node, n, with maximum level that belongs to $K_i \bigcap K_j$. The difference between level of n and levels of po_i and po_j is used to estimate the desired distance between po_i and po_j. The idea is that if this difference is small, the two outputs should be placed near one another. If $K_i \bigcap K_j$ is null, then the two outputs can be placed anywhere with respect to one another.

After assigning initial positions to the output pads, we proceed to assign the input pads. We transform the problem to a linear assignment problem. The primary inputs and outputs of the circuit define a bipartite graph. Each vertex on the input side represents a primary input. Each vertex on the output side represents a primary output. There is an edge between pi_i and po_j if and only if there exists a directed path from pi_i to po_j. The weight of edge ij is equal to the length of longest path from pi_i to po_j. (The length of a path is the number of gates on the path. Note that there may be more than one path between input-output pair.) This information is stored in an $M \times N$ matrix, T, where M is the number of primary inputs and N is the number of primary outputs.

We put S slots (as many as pad-to-pad design rules allow) at the circuit boundary and construct an $N \times S$ linear assignment cost matrix C. Entry (i, k) in matrix C represents the cost of assigning primary input

pi_i to slot s_k. This cost is calculated as follows.

$$c(i,k) = \sum_j \left(1 - \frac{d(j,k)}{t(i,j)}\right)^2$$

where $d(j,k)$ is the 'distance' between primary output po_j and slot s_k and $t(i,j)$ which represents the 'target' distance between pi_i and po_j is read from the T matrix. $d(j,k)$ must be stated in terms of number of gates. From the technology file and the net list (or a canonical mapping of the Boolean network), the average gate width, W_{avg}, is calculated. In a standard cell design environment, cell height, H, is constant and ratio of channel height to cell height, e.g., γ can be estimated. Hence,

$$d(j,k) = \frac{|x_{po_j} - x_{t_k}|}{W_{avg}} + \frac{|y_{po_j} - y_{t_k}|}{(1+\gamma) \times H}.$$

After running the linear assignment solver on matrix C, we obtain a minimum sum-cost solution to the input pad assignment problem, i.e., a subset X of entries c^{pq} of matrix C is chosen such that the following holds:

$$\forall i \ \exists j^* : \ c^{ij^*} \in X,$$
$$if \ i_1 \neq i_2 \ then \ j_1^* \neq j_2^*,$$
$$\sum_i c^{ij^*} \ is \ minimum.$$

Since rows in the cost matrix C correspond to floating input pads and columns correspond to the slots, the linear assignment determines input pad assignment with the minimum cost.

In order to assure that primary inputs which are connected to the same gates are assigned positions near each other, we cluster these inputs together and assign input clusters to slots (with capacities bigger than one). In particular, primary inputs which connect to the same gates within a level distance of l are clustered together. (Note that primary inputs are at level 0 and l is set to be a small fraction of L, the maximum level distance from any primary input to any primary output.)

In order to improve on the pin assignment result, we repeat the same procedure for the second time; this time fixing the primary inputs in their assigned slots and placing primary outputs with respect to them.

This technique can be easily extended to timing-driven IO pad placement. In that case, detailed information (intrinsic delay, output drive, input pin capacitance) about the library gates can be used to estimate the maximum path delay from any input to any output. The timing slack on each path is used to estimate the wire length that can be used on that path. Let $b(i,j)$ be the difference between the required time at po_j and the signal arrival time at primary input pi_i. It represents the path delay upper bound. Let $t(i,j)$ denote the difference between $b(i,j)$ and the maximum path delay from po_i to po_j. (Note that this path delay only includes the gate delays.) It represents the maximum propagation delay that can be 'used-up' in the wires connecting gates on that path without violating the path delay constraints. Let $d(j,k)$ be the difference between distance from output po_j to slot s_k and the total gate width on the path. It is translated to units of delay by using the capacitance per unit interconnection length which is available in the technology file. Then,

$$c(i,k) = \begin{cases} \sum_j \left|\frac{d(j,k)}{t(i,j)}\right| & \text{if } d(j,k) \leq t(i,j) \\ \infty & \text{otherwise} \end{cases}$$

Example	Total Net Length		% Improvement
	Optimal Pads	Random Pads	over Random
C1355	113205	133278	15.0
C432	88923	91557	2.8
C499	113210	133363	15.1
C880	142912	152118	6.0
alu4	38946	45768	15.2
apex7	76191	87166	12.6
b9	41503	46400	10.8
duke2	148557	155823	4.5
sa02	33512	33941	1.1

Table 2.1: Results for I/O PAD Assignment

The same initial input clustering and output pad distribution followed by linear assignment can be used to solve the timing-driven pad placement.

2.3 Experimental Results

The proposed technique has been implemented and run on several MCNC benchmark circuits. Table 1 compares our pad placement results with those of a random pad placement procedure. All data are extracted after detailed placement. We observe about 10% improvement in total wire length due to our pad assignment procedure.

2.4 Conclusions

We have presented an IO pad placement technique for assigning IO pads based on analysis of the circuit structure and path delay constraints. Both of these considerations are transformed into proximity relationships among the off-chip IOs. A cost function which penalizes violations of these proximities is defined and linear assignment technique is used to simultaneously assign IO pads to slots. This technique is general and can handle pad placement prior to technology mapping or detailed placement procedures.

Acknowledgements The authors would like to acknowledge the support received for this research from SRC Grant 90-DC-008.

Bibliography

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

[2] 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, 1982, pp.645-650.

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

[4] 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, 1988, pp. 229-259.

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

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

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

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

[9] 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., 1990, pp. 79-97.

[10] 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* to appear Jan. 1991.

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

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

[13] 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 (1979), pp. 19-36.

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

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

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