AN INTEGRATED APPROACH TO LOGIC SYNTHESIS AND PHYSICAL DESIGN

by

Massoud Pedram

Memorandum No. UCB/ERL M91/69

14 August 1991

AN INTEGRATED APPROACH TO LOGIC SYNTHESIS AND PHYSICAL DESIGN

by

Massoud Pedram

Memorandum No. UCB/ERL M91/69

14 August 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AN INTEGRATED APPROACH TO LOGIC SYNTHESIS AND PHYSICAL DESIGN

by

Massoud Pedram

Memorandum No. UCB/ERL M91/69

14 August 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

An Integrated Approach to Logic Synthesis and Physical Design

Massoud Pedram

University of California
Berkeley, California

Department of Electrical Engineering
and Computer Science

Abstract

Advances in VLSI technology have created a situation in which the timing behavior of circuits has become dominated by the interconnect delays instead of the switching delays of the cells. Efforts to cope with this situation, may be classified as follows. Some researchers incorporate timing requirements into the physical design; others perform logic restructuring, retiming and buffering in order to optimize circuit speed prior to physical design. Neither approach is adequate: timing-driven physical design, although very useful, is not able to correct all the timing problems since major characteristics of designs are dictated at the behavioral / logic level and physical design is too far down in the design pipeline to meet the performance specifications by itself. Similarly, current synthesis systems lack detailed information about factors such as layout, interconnect and characteristics of macrocell- or gate-level implementations. It is, therefore, necessary to integrate logic synthesis and physical design in order to address the requirements of today's high-density and high-performance designs.

This thesis describes an integrated approach to logic synthesis and physical design which finds solutions in both domains of design representation simultaneously and interactively. The two processes are performed in lock-step: performance analysis, logic structuring, and technology mapping take advantage of detailed information about the interconnect delays and the layout cost of various optimization alternatives; placement itself is guided by the evolving logic structure and accurate path-based delay traces. Such combined approach helps designers employ a complete top-down design methodology and move toward true system synthesis.

Prof. Ernest S. Kuh
Thesis Committee Chairman

An Integrated Approach to Logic Synthesis and Physical Design

Massoud Pedram

To my wife, Afsane

Acknowledgements

I am indebted to my research advisor Professor Ernest S. Kuh for his continuous support, constant encouragement, and guidance. Without his vision and help, this research would have not been possible. He has taught me the importance of discipline and originality in research.

I have learned a great deal from my association with Professors Robert Brayton, David Hodges, Chenming Hu, Richard Newton, and my academic advisor Alberto Sangiovanni-Vincentelli. I would also like to express my appreciation to Professors Jan Rabaey and Shmuel Oren for reviewing my dissertation on such a short notice and providing useful comments.

I am very fortunate to have worked with a number of people including Professors Weiming Dai, Malgorzata Marek-Sadowska, and Dr. Ulrich Lauther. I am indebted to Dr. Bryan Preas of Xerox PARC who introduced me to the field of computer-aided design and has been my good friend and mentor.

I am very grateful to Tahani Sticpewich whose administrative help and friendly face has made the last four years of my academic life simpler and more enjoyable.

My friendship and collaboration with Narasimha Bhat has been a great source of joy and intellectual stimulation for me. I value his good heart and clear mind. Thanks to other members of the Layout and CAD group for providing an enjoyable and challenging working environment. In particular, I would like to thank Andrea Casotto, Kamal Chaudhary, Michael Jackson, Shen Lin, Sharad Malik, Rajeev Murgai, Hamid Savoj, Minshine Shih, Kanwar Jit Singh, Arvind Srinivasan, Ren-Song Tsay and Deborah Wang.

I have benefited from interaction and joint research with our industrial visitors, most of all Stefan Mayrhofer and Yasushi Ogawa. Thanks to George Carvalho, Benjamin Chen and Charles Hough who have helped me with the programming tasks.

I am grateful to my father and mother who encouraged me to pursue academics and who taught me the value of hard work and perseverance. Special thanks to my good friends of many years, Saiid Eskandari, Bahman Salehi, and Hamid Savoj.

I would also like to acknowledge the Semiconductor Research Corporation and the National Science Foundation which supported the research work presented in this dissertation.

Above all, I am most grateful to my wife and best friend, Afsane, for her love, understanding, and faith during the last five years and for coping with the hardships of graduate student life. I would have not succeeded without her help and support. She has my everlasting love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 CAD for VLSI Design

In order to cope with the ever growing complexity of VLSI systems, engineers and designers are relying more and more on computer-aided design systems. The design of electronic systems is often divided into four stages. The high level synthesis starts with an abstract behavioral specification of a digital system and finds a register-transfer level structure that realizes the given behavior. The logic synthesis generates a net list of gates in a given technology that realizes the register-transfer level description. The layout design provides physical realization of the net lists on a chip and includes not only placement and routing but also geometric artwork. Finally, fabrication and test consists of mask-making, fabrication and performance test.

This division into design steps has been perceived as necessary in order to make the design process tractable. It is, however, artificial and restrictive. There is evidence that detailed consideration of physical issues during synthesis may lead to completely different solutions with higher quality. It is, therefore, necessary for synthesis systems to couple to physical design.

A principal goal of electronic design automation effort is to provide designers with the capability to employ a complete top-down design methodology, compiling abstract architectural

descriptions into an optimal implementation for a specific physical media. In fact, as density increases, system and ASIC designers have no choice but to converge to such a top-down design methodology. However, there is at least one major difficulty that must be overcome in order to develop the full potential and promise of this methodology. The problem is that the top-down design approach makes high-level decisions about optimization, scheduling, allocation, logic partitioning and restructuring, and so forth in terms of abstract views of behavior and structure. During these early steps, factors such as characteristics of the physical media, layout, interconnect and parasitics are ignored. Physical design which is expected to address these issues comes much later in the design hierarchy. By then, many of the key architectural and structural decisions have been made, hence, limiting capability of the physical design tools to generate the "best" solutions in terms of area and performance.

This dissertation provides appropriate models, algorithms and techniques to increase the power and robustness of the top-down design methodologies. The goal – which is easy to state but very difficult to achieve – is to *integrate* various design steps into a single step while *managing* the computational complexity. The first part of the dissertation describes models and techniques to couple logic synthesis to placement and routing while the second part gives details of a hierarchical floorplanning approach that combines the placement, shape assignment, global routing and pin assignment steps for macrocell layout.

Logic synthesis generally involves two distinct phases. The first phase performs technology-independent transformations on the input logic equations. The second phase performs technology-dependent mapping and optimization to implement the logic equations using gates in a target library. Most previous work [Brayton 87b, Hachtel 88] has focused on minimizing gate area and delay through a chain of gates without considering the area needed to hold the interconnect lines and the delay through the lines. It is generally assumed that interconnect optimization can be relegated to the physical design phase. However, as the packing density of transistors and their intrinsic gate delay improve with technological advances, on-chip interconnect becomes a major factor in determining the speed, power consumption and size of circuits. At the same time, physical design cannot correct many of the timing problems which arise due to decisions made higher up in the design hierarchy. Therefore, it is necessary to develop models and algorithms for explicitly controlling interconnect during synthesis.

In the first part of this dissertation, effects of interconnect on circuit area and performance are analyzed; appropriate models and computational procedures for the accurate estimation of wiring delays during synthesis are presented; algorithms for doing the I/O pad assignment and placement of Boolean networks (which are directed acyclic graphs representing input logic equations) are described; techniques for coupling logic synthesis to placement and for maintaining the correspondence between data representations in logic and layout domains are introduced; and finally new methods for common factor extraction, gate matching and fanout optimizations are introduced which aim at minimizing the interconnect and routing density as well as the active gate area and delay.

It is very difficult to predict the performance of submicron silicon before layout. High-level delay models which ignore (or oversimplify) the interconnect and layout effects are the state of art. More elaborate methods for accurate calculation, modeling and simulation of timing delays, which are at this time dominated by metal interconnect rather than gate delays, are needed. Floorplanning tools can provide accurate delay estimations for pre-layout simulation. At the same time, they can give system designers an opportunity to interface with layout tools to reduce metal interconnect delays. Designers can optimize critical paths by physically placing macros or logical blocks before layout. Floorplanning is, therefore, another way to provide high level design tools with detailed information about module sizes and shapes, interconnect delays, and critical sections of the design.

In the early 80's, several authors found out that the sizing of slicing floorplans with variable-shape cells can be done efficiently [Stockmeyer 83, Otten 83]. Soon, floorplanning systems emerged that combined partitioning methods known from placement with the efficient shape computation method [La Potin 86, Zimmerman 86]. These systems were quite successful since they could handle flexible macros effectively and use little run time. This work was later extended to incorporate the wiring area estimation [Zimmerman 88, Dai 89] and the orientation of cut lines during the shape computation [Zimmerman 88]. At the same time, global wiring was integrated into the floorplanning process by [Dai 87c].

In the second part of this dissertation, a hierarchical floorplanning based on cluster tree generation, shape computation and floorplan optimization techniques is presented. The floorplanner minimizes chip area and total interconnection length subject to various topological, geometrical and timing constraints. Novel features of the floorplanner may be stated

as: extension of shape computation method to multi-way unoriented clusters; a systematic top-down floorplan optimization which minimizes wire length as well as chip area; efficient and accurate wiring area estimation methods during the shape computation and floorplan optimization; integration of pin assignment and global routing procedures; introduction of net-based and path-based performance-oriented floorplanning techniques; and accommodation of analog device constraints. The floorplanner can be used either in the feedback loop of a high-level synthesis system or as a stand-alone system for final layout generation in macrocell design.

1.2 Overview

Chapter 2 introduces LILY [Pedram 91a, Pedram 91c], a logic synthesis program, built on top of MIS [Brayton 87b], which is tightly and interactively coupled to placement tools and is capable of making synthesis decisions based on both logic-level information and detailed data about the interconnecting wires and characteristics of the physical media. LILY's key idea is to generate a point placement of the unoptimized multi-level Boolean network which captures structure of the network on the layout plane. This point placement, in turn, is used to guide the decomposition process and is later used to help estimate the wiring cost of a match during the mapping process.

In order to place the Boolean network, sides and relative positions of the primary inputs and outputs of the network must be known. Chapter 3 presents an algorithm for doing off-chip I/O pad assignment on a logic circuit. The technique which is based on the analysis of the circuit structure and path delay constraints, uses I/O pad clustering, goal-programming and linear-sum assignment to assign locations to I/O pads. This procedure can be invoked before or after logic synthesis.

Logic synthesis is driven by layout information derived from the placement of the Boolean network. It is, therefore, essential to generate a placement solution which not only captures the global connectivity structure of the network, but also produces the shortest directed path between any pair of primary input – primary output nodes. Chapter 4 describes a flow-oriented approach to the placement of general directed acyclic graphs.

Chapter 5 is devoted to logic decomposition and technology mapping. The major issue in decomposition is the identification of frequently used common subexpressions. Sharing of such expressions across the entire design reduces complexity of the synthesized network. Previous approaches select a multiple cube factor which provides the greatest cost reduction in terms of the number of literals. Here, a decomposition procedure with the objective of minimizing interconnections in the synthesized network is described. Next, a procedure for technology decomposition (i.e., converting an optimized Boolean network into an equivalent network restricted to a set of base functions such as two-input NAND gate and inverter) is presented. The objective is to find a decomposition which provides a good starting point for the layout-oriented technology mapping. In particular, logic function associated with each node in the Boolean network must be decomposed such that signals coming from nearby regions of the network enter the decomposition tree for the node at topologically near points. Finally, a technology mapping program which uses information derived from a dynamically updated global placement in order to produce more wirable circuits with lower post-layout area and higher performance is presented. The algorithm is based on DAG covering and integrates gate placement and global route modeling with a dynamic programming procedure.

Chapter 6 introduces BEAR-FP [Pedram 90c], a macrocell layout system which builds on its predecessor, the BEAR system [Dai 87d]. It inherits BEAR's dynamic and efficient data representation, which unifies topological and geometrical information, and BEAR's routing system. BEAR-FP, however, has a new floorplanning procedure with integrated global routing and hierarchical pin assignment, a new Steiner-tree global router, a detailed channel pin arrangement procedure, and a timing-driven clustering and placement capability.

Chapter 7 describes a floorplanning procedure whose objective is to assign positions, shapes and pin locations to a set of macrocells minimizing chip area and interconnection length subject to a variety of constraints. The chapter begins by an extensive review of related work followed by detailed descriptions of cluster tree generation, shape computation and floorplan optimization steps. Next, a novel technique for hierarchical pin assignment with integrated global routing is presented. The channel pin arrangement problem and some analog floorplanning issues are discussed last.

Performance-oriented floorplanning is the subject of Chapter 8. Initially, a net-based ap-

proach is presented which transforms path delay constraints into upper bound net length constraints and modifies the floorplanning steps in order to minimize violations of these constraints. Next, a mathematical programming approach for solving the initial placement of macrocells is described and techniques for solving the problem (which in the presence of non-overlap constraints is a non-convex non-linear programming problem) are discussed. Based on the coordinates of the macrocells obtained from this placement, a cluster tree is constructed and subsequently floorplanned.

Chapter 9 is devoted to an interconnection analysis tool targeting standard-cell assemblies which produces accurate pre-layout area and wire length estimates for random logic. This is important since good physical design of large systems requires accurate size estimates of the individual macrocells for area planning, optimal placement, and routability predictions, and macrocells are at times laid out using standard-cell or gate-array styles. Wire length estimates could be very beneficial to high-level and logic synthesis tools as well. A model which abstracts the important features of placement, global routing and channel routing is described. The predicted results are obtained from analysis of the net list. No prior knowledge of the functionality of the design is used. Chapter 10 contains concluding remarks and future research directions.

Part I

Logic Synthesis

Chapter 2

System Overview

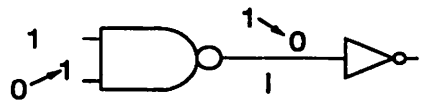
2.1 Motivation

Logic synthesis and physical design are of pivotal importance in the computer-aided design of integrated circuits. Very efficient and sophisticated techniques for the synthesis of logic circuits and placement and routing of gate net lists exist. However, to the best of my knowledge, no successful attempt has been made to integrate the two processes. Most CAD systems do structured design of electronic systems using a top-down design methodology where logic synthesis is followed by physical design. However, decisions during the logic synthesis (including partitioning logic circuits into an interconnection of combinational logic components and registers, optimizing logic, binding optimized logic equations into gates in a target library) are difficult because their effects on the final layout are hard to predict and may not become apparent until much later in the design process. Once any parameter at the layout design stage fails to satisfy a constraint imposed on it, the logic synthesis must be modified (even repeated) so as to accommodate the constraint. (This may become necessary because physical design is too far down in the design pipeline to solve performance issues that were not considered earlier.) The new change may cause some other constraint to be violated and the process must be repeated. In addition, there is evidence that consideration of physical design issues can cause one to choose a completely different gate-level implementation for a given RTL specification.

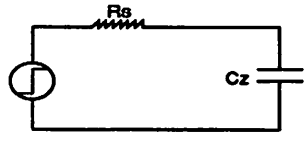
The gap between synthesis and physical design is perhaps the biggest problem ASIC designers face today, according to Peter Hsieh, co-founder and president of Elite Microelectronics (San Jose, CA), a company that designs high performance ASICs for PCs and workstations [Tuck 90]. In order to benefit from a complete top-down design methodology, to handle the ASIC designs of near future with more than 100K gates, reduce the number of design iterations, and find better quality solutions in shorter cpu time, physical design must be integrated with logic synthesis. This will allow physical design and logic synthesis to evolve together and will set the stage for true system level integration.

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 has led 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 synthesis.

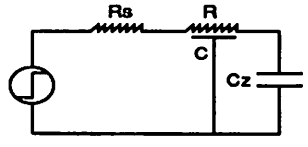
Interconnections are becoming a major concern in today's high-performance, high-density ASIC designs because the distributed RC time delay of these lines increases rapidly as chip sizes grow and minimum feature sizes shrink [Bakoglu 86]. With recent studies [Saraswat 82, El-Mansy 88] indicating that interconnections occupy more than half the total chip area and account for a significant part of the chip delay, it is appropriate that wiring is integrated into the cost function for logic synthesis. To elaborate on this point, consider Figure 2.1 which shows a performance-optimized two-input NAND gate driving a performance-optimized inverter gate through 0.2 cm of aluminum interconnect (2 μm wide, 0.5 μm thick, with a 1.0 μm thick field oxide beneath it). 0.2 cm is the expected length of a *local* interconnect line on a $2cm \times 2cm$ chip [Bakoglu 86]. Two methods are used to calculate the rise time (to 50% of its final value) at the input of the inverter gate: one method ignores the capacitance and resistance of the interconnect line, the second method accounts for them. Gate delays are taken from data sheets for an industrial 1-micron ASIC library; interconnect capacitance and resistance are calculated using expressions given in [Bakoglu 86]. The delay calculations clearly show that interconnect capacitance dominates gate input capacitance



τ = 0.3 ns R = 300 Ω /cm
Rs = 1 KΩ C = 2.5 pF/cm
Cz = 0.1 pF l = 0.2 cm



$d(z) = \tau + Rs\ Cz = 0.4$ ns



$$d(z) = \tau + Rs\ Cz + Rs\ C\ l + R\ Cz\ l + 0.5\ R\ C\ l^2$$
$$= 0.921 \text{ ns}$$

Figure 2.1: Comparison of interconnect delay to gate delay for 1-micron ASIC library

and interconnect resistance may be ignored without introducing much error.[1] Therefore, an accurate expression for propagation delay through gates connected by local interconnect lines is given by

$$d(z) = \tau + R_s(C_z + Cl)$$

where τ is the intrinsic gate delay, R_s is the on-resistance of the driver gate, C_z is the input capacitance of the fanout gate, C is the interconnect capacitance per unit length and l is the interconnect length.

In summary, with the existing technology, the capacitive term is dominated by the capacitance between the interconnection and substrate. For local aluminum lines, the resistive term is dominated by the on-resistance of the MOS transistor; for polysilicon and global aluminum lines on large-size circuits, the resistive term is controlled by the interconnection resistance. As the chip dimension increases and the minimum feature size decreases, the

[1] When the RC tree forms branches, delays for the branching nodes can be calculated independently and accumulated to obtain the delays at the sink nodes. This calculation, however, requires knowledge of net topologies which is not available before global routing. This effect will not be addressed here. Furthermore, the transmission line properties of interconnect lines are ignored for on-chip connections.

interconnection resistance increases rapidly while the MOS on-resistance remains relatively unchanged; the interconnection capacitance bottoms at about 1 - 2 pF/cm while the input gate capacitance decreases. Therefore, the distributed RC delay of interconnect lines will become even more dominant in the future.

2.2 LILY: Layout Integrated Logic Synthesis

LILY provides a framework for interaction between physical design and logic synthesis. The defining feature of LILY is its ability to incorporate layout considerations into logic synthesis and to provide a more uniform design flow from synthesis down to layout. LILY's principal idea is to generate a placement of the unoptimized multi-level Boolean network which captures structure of the network. The placement is either incrementally (during decomposition) or dynamically (during technology mapping) updated in order to maintain the correspondence between logic and layout representations. The placement information is used to evaluate the cost of a kernel extraction or a gate matching during decomposition and mapping processes.

In the end, a synthesized network along with a "companion" placement solution are generated. The placement solution is then globally relaxed in order to produce a feasible placement according to the target layout style (e.g., standard-cell or sea-of-gates). This design flow is in contrast with the existing synthesis systems which separate logic and layout optimizations.

Chapter 3

I/O Pad Assignment for the Boolean Network

3.1 Background

The key to incorporating layout aspects into logic synthesis is the generation of a placement of the multi-level Boolean network which is subsequently used to guide the decomposition and mapping processes. (See Chapter 5.) However, when using force-directed and mathematical programming approaches for solving the placement problem, positions of the off-chip I/O pads must be known prior to placing the gates [Tsay 88, Kleinhans 90, Srinivasan 91]. This is because in the absence of off-chip I/O pads, the gates collapse to the center of chip. At the same time, different I/O pad assignments give rise to placements with vastly different qualities. In particular, significant improvements in area and wire length can be obtained by doing a good I/O pad assignment.

One approach is to treat I/Os as floating gates and use a force-directed approach to assign positions to all gates. I/Os are later assigned to fixed pads [Wipfler 85]. The problem with this approach is that during the placement phase, I/Os are allowed to float (and hence assume infeasible positions) and, therefore, when they are moved to fixed pad positions, the quality of placement solution becomes questionable. This problem is more severe if all pads are constrained to be at the chip boundary. Another common approach is to use an

arbitrary I/O pad assignment prior to placement and then re-position pads based on the detailed placement result. The two phases may be iterated until an acceptable placement solution is generated. This approach is also undesirable since even if convergence is achieved, the final solution is heavily influenced by the initial pad assignment which was arbitrary. In addition, the iteration process is costly and time-consuming.

The I/O pad assignment becomes more important if there are path-delay constraints from primary inputs to primary outputs. In that case, initial locations of the I/O pads will greatly influence the quality of timing driven placement obtained. In particular, a poor pad assignment may result in an infeasible placement solution.

In the following sections, an I/O pad assignment procedure which is based on the analysis of circuit structure (in logic equation or directed net list forms) and path-delay constraints is presented [Pedram 91b]. This procedure can be invoked prior to logic optimization or final gate placement. In either case, the pad assignment result is used as input to placement tools. The resulting placement can then be used either to guide logic decomposition/restructuring and technology mapping procedures or is followed by routing to generate the final layout.[1]

The I/O pad assignment technique can be summarized as follows. Initially, primary outputs are ordered in a manner which maximizes the proximity between their transitive fanin cones. Then, output pads are distributed on the chip boundary and *goal distances* for primary input - primary output pairs are calculated based on the analysis of circuit structure and path-delay timing constraints. Next, a number of slots are placed on the chip periphery and primary inputs are assigned to slots such that the sum over all primary input - primary output pairs of violations of the goal distances is minimized. Linear-sum assignment technique is used in order to solve this problem efficiently and concurrently. In order to assure that primary inputs which are connected to the same gates are assigned positions near each other, inputs are divided into groups and each group is assigned to a set of adjacent slots.

[1]The I/O pad assignment may be known from the results of earlier design steps. For example, an early floorplanning solution may have dictated the positions of I/Os based on global system considerations. (See Chapter 7.) In such a case, there is no need for the bottom-up pin assignment procedure.

3.2 Terminology

The circuit is specified in the form of a directed acyclic graph (DAG), that is, a *Boolean network* prior to logic optimization or a *directed net list* prior to gate placement. Therefore, I shall give some terminology and definitions relevant to directed graphs followed by statement of assumptions and timing model used.

A path in a directed graph is an open walk with no repeated vertices which follows the edge orientations. A (u, v) path is a path with u and v as end vertices. A (u, v) *bidirected* path consists of exactly two sub-paths (either $((u, z) \wedge (v, z))$ or $((z, u) \wedge (z, v))$) for some vertex z. Distance $d(u, v)$ is the minimum number of edges in a (u, v) path. Bidirected distance $b(u, v)$ is the minimum number of edges in a (u, v) bidirected path. In both cases, if there is no such path, distance is set to ∞.

A node u is a *fanin* of a node v if there is a directed edge e_{uv} from u to v and a *fanout* if there is a directed edge e_{vu}. A node u is a *transitive fanin* of a node v if there is a (directed) path from u to v and a *transitive fanout* if there is a (directed) path from v to u. *Primary inputs* are inputs of the directed graph and *primary outputs* are its outputs. *Internal nodes* are nodes of the directed graph with at least one fanin and one fanout. The *primary input support* of a node u is the set of primary inputs that are transitive fanins of u.

Each internal node has an exact (in case of gates in a net list) or estimated (in case of unmapped nodes of a Boolean network) area. The average dimensions of an internal node can, therefore, be calculated.

Consider a node u and let v be its fanout node. The delay through u for a signal transition at one of its inputs is given as

$$\tau + R\,(C_{gate} + C_{wire})$$

where τ is the intrinsic delay through u, R is the output resistance of u, C_{gate} is the input capacitance of v and C_{wire} is the lumped capacitance of line e_{uv} which is given by

$$C_{wire} = C_H\,|x_u - x_v| + C_V\,|y_u - y_v|.$$

C_H and C_V denote the capacitances per unit length of horizontal and vertical interconnect wires respectively and (x_u, y_u) and (x_v, y_v) denote positions of nodes u and v.

If node is a gate in the library, its intrinsic delay, input pin capacitance, and output resistance are known. Otherwise, they can be estimated. One such estimation technique is described in [Wallace 90]. That paper presents a simple model for estimating the delay of a multi-level combinational logic description prior to technology-dependent mapping. The model proposes that delay through a node varies logarithmically with both the complexity and fanout of the node's logic equation. The input pin capacitance for a node is taken to be equal to that of a two-input NAND gate in the target library and the drive capability for the output pin is set to that of an inverter.

3.3 The Basic Approach

For each primary output po_i, the circuit is traversed in a depth-first order and primary input supports are identified. Then, a linear ordering on the primary outputs maximizing proximity among their primary input supports is derived. This is accomplished by creating a block B_i for each primary output po_i and creating a pin p_j on the block corresponding to each primary input pi_j in the primary input support of po_i. Primary inputs which belong to the primary input support of exactly one primary output are ignored since they do not affect the proximity metric for primary outputs. A linear placement of blocks B_i which minimizes the total net span is generated. This solution corresponds to a linear ordering on the primary outputs which maximizes proximity among their primary input supports. (See Figure 3.1.)

Bidirected distances between consecutive pairs in the ordered list of primary outputs are calculated by performing depth first search from outputs toward the inputs. The total bidirected distance from the leftmost to the rightmost primary output in the ordered list is normalized to chip boundary length and the primary outputs are distributed on the chip periphery accordingly. The idea is that if the bidirected distance between a pair of outputs is small, the two outputs should be placed near one another. If the bidirected distance between a pair of outputs is ∞, then the two outputs can be placed anywhere with respect to one another. In particular, they are placed near one another.

After assigning initial positions to output pads, input pads are assigned. Let M denote the number of primary inputs in the circuit. The pad placement problem is transformed

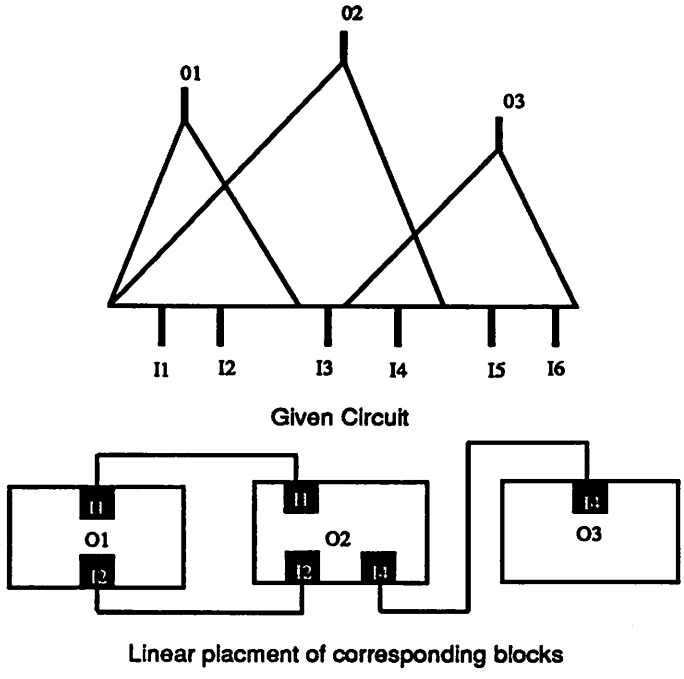


Figure 3.1: Output pad ordering based on linear placement

into a linear-sum assignment problem. S slots are placed on the circuit boundary and an $M \times S$ linear assignment cost matrix $[C]$ is constructed. (S is equal to θ times the number of floating I/Os where $\theta \geq 1.0$.) Entry (i,k) in matrix $[C]$ represents the cost of assigning primary input pi_i to slot s_k. This cost is calculated as

$$c(i,k) = \sum_{j \in po's} \left(1 - \frac{h(j,k)}{d(i,j)}\right)^2$$

where $h(j,k)$ is the half perimeter length of the bounding box enclosing primary output po_j and slot s_k. $d(i,j)$ is equal to the distance from pi_i to po_j. It has been converted into Manhattan length using the average dimensions for an internal node.

After running a linear assignment algorithm [Burkhard 80] on matrix $[C]$, a minimum sum-cost solution to the input pad assignment problem is obtained, i.e., a subset X of entries c_{pq} of matrix $[C]$ is chosen such that the following holds

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

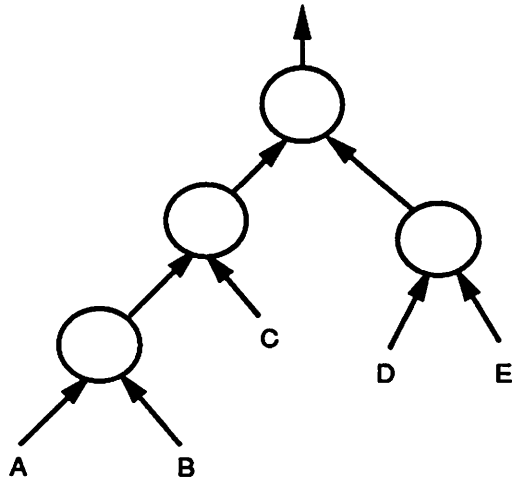


Figure 3.2: Input pad clustering based on bidirected distances

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 ensure that primary inputs which are connected to the same gates are assigned positions near each other, these inputs are clustered together and clusters are assigned to adjacent slots during the linear assignment step. In particular, primary inputs whose pairwise bidirected distances are less than or equal to l are grouped. (l is set to be a small fraction of L, the number of levels in the DAG.) For example, in Figure 3.2, inputs A through E will be clustered together for $l \geq 5$.

3.4 The Timing Driven Approach

Required times at the primary outputs and arrival times at the primary inputs of the functional block are given. The timing slack on each path is used to estimate the wire length that can be accommodated on that path as described below.

Let $a(i,j)$ be the difference between the required time at po_j and the signal arrival time at primary input pi_i (i.e., the allowed path delay) and $g(i,j)$ be the longest path delay

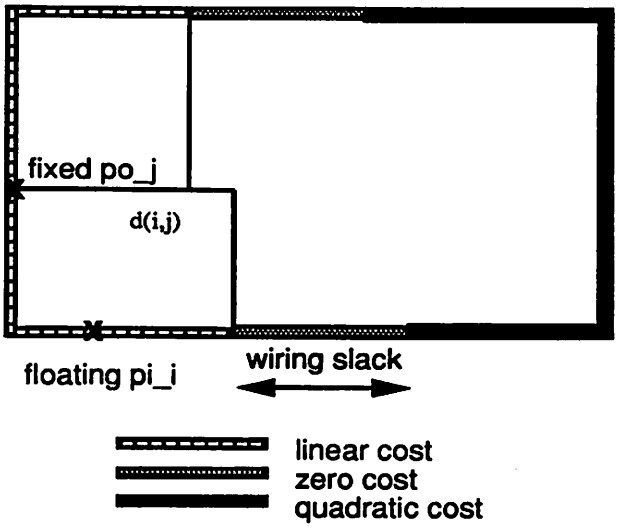


Figure 3.3: Cost function for timing-driven I/O pad assignment

from pi_i to po_j calculated recursively as in [Hitchcock 82]. $g(i,j)$ does not include the wiring delay contribution, that is, $C_{wire} = 0$ in the delay equation. Now, let $w(i,j) = a(i,j) - g(i,j)$ represent the maximum delay that can be allocated to signal propagation through wires connecting gates (which lie on paths from pi_i to po_j) without violating the timing constraints. This delay is translated to a Manhattan length by using the values of C_H and C_V from the technology file.

Let $h(j,k)$ be the half perimeter length of the bounding box enclosing primary output po_j and slot s_k and $d(i,j)$ be the distance from input pi_i to output po_j. $d(i,j)$ is converted to units of Manhattan length as in Section 3.3. Then, the cost function is defined as (Figure 3.3)

$$c(i,k) = \sum_{j \in po's} t(i,j,k)$$

$$t(i,j,k) = \begin{cases} d(i,j) - h(j,k) & \text{if } h(j,k) \leq d(i,j) \\ 0 & \text{else if } d(i,j) < h(j,k) \leq d(i,j) + w(i,j) \\ (h(j,k) - d(i,j))^2 & \text{otherwise} \end{cases}$$

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

3.5 Experimental Results

example	# gates	# inputs	# outputs
C1355	210	41	32
C1908	244	33	25
C3540	589	50	22
C432	127	36	7
C5315	588	178	123
C880	221	60	26
bw	93	5	28
duke2	235	22	29
e64	185	65	65
misex2	60	25	18
misex3	300	14	14
rd84	87	8	4

Table 3.1: Example circuits (# gates after mapping)

The techniques described here have been implemented in a computer program named PACT (Pad Assignment based on Circuit sTructure). PACT is written in C and has been incorporated into LILY. PACT was run on several MCNC logic circuit benchmarks [MCNC 88] (after logic optimization and technology mapping). Table 3.1 shows some characteristics of the benchmarks. PACT results were compared with those of *random* and *clockwise* I/O pad assignment procedures provided in Octtools release 5.0. The first procedure randomly assigns a side and position along the side to each pad. It was repeated 100 times for each example with different seeds and the average wire lengths are tabulated. The second procedure selects an output pad and assigns that output and all primary inputs in its support set to the pads around the chip boundary in a clockwise fashion. It then processes the next output and so on. The procedure is very sensitive to the order in which outputs are picked. The clockwise procedure was run 20 times with different output orderings and average wire lengths are reported.

The tabulated data are collected after pad assignment, detailed placement by GORDIAN placement package [Kleinhans 90], and global and detailed routing tools of Octtools 5.0. The area, wire length and worst case path delay are based on the placed and routed circuits and include the delay through interconnect lines. The circuits were optimized and mapped using MIS-II [Brayton 87b]. A library similar to MCNC standard cell library [Heinbuch 88]

example	total net length			% improvement over	
	random	clockwise	PACT	random	clockwise
C1355	210.6	203.8	184.5	12.4	9.8
C1908	248.2	238.1	224.0	9.8	5.9
C3540	1215.2	1135.4	1047.0	13.8	7.8
C432	112.0	100.8	88.3	21.2	11.6
C5315	1823.2	1716.3	1423.3	21.9	17.0
C880	252.3	227.1	192.8	23.6	15.1
bw	71.2	66.1	63.5	10.8	3.9
duke2	339.5	326.0	300.4	11.4	7.9
e64	152.7	139.2	121.2	20.6	12.9
misex2	41.9	37.6	35.1	16.2	6.7
misex3	459.4	433.0	415.7	9.5	4.0
rd84	52.7	51.2	49.3	6.5	2.0

Table 3.2: Wiring Results for I/O PAD Assignment (wire lengths in millimeters)

with modified timing parameters was used so that these parameters match those of a realistic 1 micron library. ($C_H = C_V = 2.5pF/cm.$) Table 3.2 shows the total interconnection of the benchmark circuits. Average reductions of 15.1% and 8.4% over random and clockwise procedures were observed.

Results showing effect of the I/O pad assignment on the circuit speed are tabulated in Table 3.3. Average improvements of 4.1% and 3.1% over random and clockwise procedures were observed. Note that the GORDIAN placement package does not have a timing-driven capability, therefore, Table 3.3 does not truly reflect the impact of I/O pad assignment on the circuit delay. Furthermore, the drive capability of a pad is much higher than that of a gate, and a good placement tool can reduce the path delay by allowing pads to drive longer wires on critical paths. PACT run times are short (e.g., 8 seconds output ordering, 33 seconds linear assignment for C880 on a DEC3100 workstation).

example	delay without wiring	delay with wiring			% Improvement over	
		random	clockwise	PACT	random	clockwise
C1355	13.6	17.5	17.2	15.9	9.7	7.6
C1908	20.3	25.6	25.4	24.5	4.3	3.5
C3540	29.9	39.1	38.7	38.1	2.6	1.5
C432	21.6	25.8	25.3	25.2	2.4	0.4
C5315	20.1	27.1	26.3	26.2	3.3	0.4
C880	23.5	28.2	27.8	26.2	7.1	5.8
bw	23.9	27.4	27.8	27.1	1.0	2.5
duke2	18.4	24.2	24.0	23.3	3.7	2.9
e64	41.8	46.1	46.4	45.0	2.4	3.0
misex2	7.5	8.2	8.2	8.1	1.2	1.2
misex3	16.4	22.9	22.8	21.1	7.9	7.5
rd84	11.1	12.6	12.3	12.2	3.2	0.8

Table 3.3: Timing Results for I/O PAD Assignment (delays in nano-seconds)

Chapter 4

Placement of the Boolean Network

4.1 Background

Automatic placement for VLSI chip design has been extensively investigated in the past. Most of the approaches proposed so far are based on one of the following optimization techniques: min-cut bi-partitioning [Breuer 77, Lauther 79, Mayrhofer 90], simulated annealing [Sechen 88], or quadratic optimization [Antreich 82, Cheng 84]. While placement algorithms based on min-cut bi-partitioning are very fast, the quality of the final solution may vary drastically. Simulated annealing gives excellent solutions at the expense of long computation times. More recently, a placement algorithm based on quadratic optimization has been proposed where the uniform distribution of the modules over the placement area is achieved by alternating global optimization and bi-partitioning steps over several levels of hierarchy. From the module coordinates calculated during the global optimization steps, initial bi-partitions are derived which are then improved using the Fiduccia-Mattheyses heuristic [Fiduccia 82]. Here, an extension of this approach which is suitable for partitioning Boolean networks is introduced.

It is very important that the placement solution reflects not only the global connectivity of the Boolean network but also produces directed paths between any pair of primary input

and primary output nodes which do not zigzag or meander outside the pair's bounding box. Therefore, the netlist-oriented view of the placement problem must be replaced by a flow-oriented view [Mayrhofer 91]. In particular, a degree of freedom which has not been used by previous partitioning algorithms is to be exploited. To develop this observation, consider Figure 4.1 which depicts a very small Boolean network together with two bi-partitioning solutions. These solutions are equivalent in terms of the number of edges of the DAG

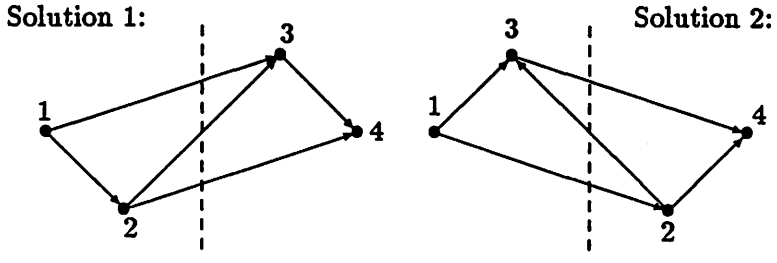


Figure 4.1: Comparison of two bi-partitionings of a DAG

crossing the cut line. While for the first partitioning solution, every path from module 1 to module 4 crosses the cut line at most once, in the second partitioning solution path $\{1,2,3,4\}$ crosses the cut line three times. The first solution is more desirable (at least from a timing viewpoint). Consequently, during the bi-partitioning step both the number and direction of edges crossing the cut line should be considered.

4.2 Terminology

Given a set M of modules and a set N of nets, a netlist specifies all modules M_ν connected by the net $\nu \in N$. For Boolean networks, the source μ_ν^+ and sinks $\mu \in M_\nu^-$ for every net $\nu \in N$ must be distinguished. (See Figure 4.2.) This leads to a transformation of the netlist description into a directed acyclic graph $D = (M, E)$. The set of nodes in the DAG is identical to the set of modules. The set of edges is derived from the source-sink net model in the following way: For every net $\nu \in N$, the set of edges E_ν is defined as $E_\nu = \{(\mu_\nu^+, \mu) \mid \mu \in M_\nu^-\}$. The set of edges E of the DAG is given by $E = \cup_{\nu \in N} E_\nu$. A sequence $(\mu_i, \mu_{i+1}), (\mu_{i+1}, \mu_{i+2}), \cdots, (\mu_{j-1}, \mu_j)$ of edges $e \in E$ defines a directed path in the DAG from module μ_i to module μ_j: $p(\mu_i, \mu_j)$.

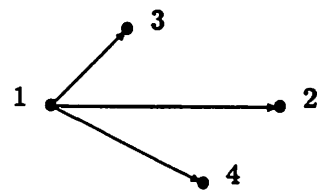


Figure 4.2: The source-sink net model

Nodes in a Boolean network belong to either the set of primary inputs I, the set of primary outputs O, or the set of internal modules G. The network can be partitioned into a set of logic cones where each logic cone corresponds to a primary output and all its transitive fanin nodes. More formally, for every primary output $o \in O$, the corresponding cone C_o is described by the set C_o^M of modules and the set of edges C_o^E characterized by

$$\begin{aligned} C_o^M &= \{\mu \in I \cup G \mid \exists p(\mu, o)\} \\ C_o^E &= \{(\mu_i, \mu_j) \in E \mid \mu_i \in C_o^M \wedge \mu_j \in C_o^M\} \\ C_o^S &= \{\mu \in I \mid \mu \in C_o^M\} \end{aligned}$$

The last equation defines the primary input support of a cone, i.e., all modules in the set $\mu \in C_o^M$ which are primary inputs.

The *preferred direction* φ_e^p for an edge $e \in E$ is the most desirable orientation from the source to the sink of the edge in any placement of the Boolean network. A placement of the Boolean network in which all edges assume (as much as possible) orientations equal to their pre-computed preferred directions and the total wire length is minimum is the objective of the flow-based placement algorithm. This is roughly equivalent to minimizing the total wire length and the number of primary input – primary output paths that zigzag or meander outside the pair's bounding box.

4.3 Computing Preferred Edge Directions

For the example given in Figure 4.1, it was obvious that the preferred direction for edges in the DAG is from left to right. Here, a general procedure for assigning a preferred direction to each edge of the Boolean network is given. The procedure *ComputePrefDirections* described

next is based on the assumption that locations (x_μ, y_μ) of all primary inputs and primary outputs $\mu \in I \cup O$ are prespecified and remain unchanged during the placement. In order

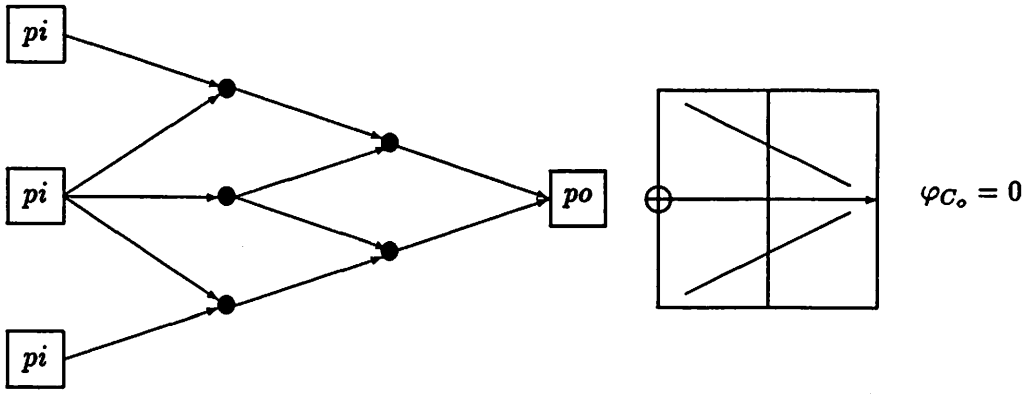


Figure 4.3: Computing direction for a cone C_o

to calculate the preferred directions for all edges $e \in E$, sets C_o^M, C_o^E, and C_o^S must be calculated for each primary output $o \in O$. Since positions of modules $\mu \in C_o^S$ in the support of a cone are prespecified, the center of mass for support C_o^S is easily calculated as

$$x_o^S = \frac{1}{\mid C_o^S \mid} \sum_{\mu \in C_o^S} x_\mu \qquad y_o^S = \frac{1}{\mid C_o^S \mid} \sum_{\mu \in C_o^S} y_\mu \quad .$$

The direction φ_{C_o} of the cone, which is defined as the orientation of the axis from the center of mass of C_o^S to the primary output $o \in O$, is determined next. All edges $e \in C_o^E$ of the cone are assigned this direction as their preferred directions φ_e^p. (See Figure 4.3.) In general, an edge may lie in several cones with different directions. In this case, the preferred direction φ_e^p is derived as the mean value of the directions of all cones in $C_e = \cup_{o \in O}\{C_o \mid e \in C_o^E\}$. The procedure can then be summarized as

Procedure ComputePrefDirections()

input: $D = (M, E)$, (x_μ, y_μ) $\forall \mu \in (I \cup O)$
output: φ_e^p $\forall e \in E$
complexity: $O(\mid O \mid \cdot \mid E \mid)$

1) **for_all** ($e \in E$) $\varphi_e^p = 0$
2) **for_all** ($o \in O$)

2.1) determine C_o^E and C_o^S
2.2) calculate x_o^S, y_o^S, and φ_{C_o}
2.3) **for_all** ($e \in C_o^E$) $\varphi_e^p = \varphi_e^p + \varphi_{C_o}$
3) **for_all** ($e \in E$) $\varphi_e^p = \varphi_e^p / \mid C_e \mid$

Since positions of primary inputs and primary outputs are prespecified, this procedure is executed once before placement. Clearly, the initialization step (1) and the calculation of the mean value in step (3) can be done in $O(\mid E \mid)$. The complexity of the procedure is determined by step (2) where for each of the $\mid O \mid$ primary outputs, a depth first search (which takes $O(\mid E \mid)$ steps [Aho 74]) is performed.

4.4 DAG Bi-partitioning

The cost function for DAG bi-partitioning is used to score various bi-partitionings of a set M of modules into two disjoint subsets A and B with $M = A \cup B$. For the Fiduccia-Mattheyses heuristic, the cost of a net $\nu \in N$ is derived from the assignment of cells $\mu \in M_\nu$ to the subsets A and B: the cost is one if $M_\nu \cap A \neq \emptyset \wedge M_\nu \cap B \neq \emptyset$ and zero otherwise.

For DAG bi-partitioning, all edges $e \in E_\nu$ on a given net must be examined. Figure 4.4 illustrates that the direction φ_e of an edge depends on both the direction of the cut line and the assignment of the net source to either subset. For a vertical (horizontal) cut, assume that the subset A is to the left (bottom) of B. If $\mu_\nu^+ \in A$, then $\varphi_e = 0$ ($\varphi_e = \frac{\pi}{2}$); else if $\mu_\nu^+ \in B$, then $\varphi_e = \pi$ ($\varphi_e = \frac{3\pi}{2}$).

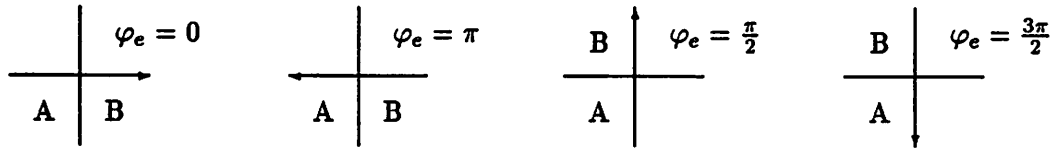
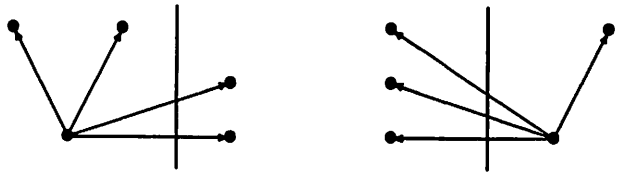


Figure 4.4: Direction of an edge crossing a cut line

For each net ν, vector $\alpha_\nu = \{aa_\nu, ab_\nu, ba_\nu, bb_\nu\}$ captures the flow of its edges across a cut line as illustrated by examples in Figure 4.5. In the first case, the net source is in subset A

while net sinks are in subsets A and B. The number of edges in subset A (aa_ν) is 2 and the number of edges crossing the cut line from left to right (ab_ν) is 2. In the second case $ba_\nu = 3$ edges cross the cut line from right to left and $bb_\nu = 1$ edge is in subset B. Since the source of a net is assigned to either A or B, two components of this vector are always zero. The cost function for the DAG bi-partitioning is expressed in terms of these vectors.



$\alpha_\nu = (2,2,0,0), \Psi(\nu) = 1$ $\alpha_\nu = (0,0,3,1), \Psi(\nu) = 2$

Figure 4.5: Examples for the vector α_ν, $\varphi_e^d = 0 \quad \forall \quad e \in E_\nu$

The basic idea is to penalize the deviation of the direction of an edge from its preferred direction by an additive term in the cost function given as

$$\Psi(\nu) = \begin{cases} 0 \ , \ \text{if} \ ab_\nu = 0 \ \wedge \ ba_\nu = 0 \\ 1 \ , \ \text{if} \ ab_\nu > 0 \ \vee \ ba_\nu > 0 \ \text{and} \ \forall_{(\mu_\nu^+,\mu)\in E_\nu} \ \mid \varphi_e - \varphi_e^p \mid < \frac{\pi}{2} \\ 2 \ , \ \text{if} \ ab_\nu > 0 \ \vee \ ba_\nu > 0 \ \text{and} \ \exists_{(\mu_\nu^+,\mu)\in E_\nu} \ \mid \varphi_e - \varphi_e^p \mid \geq \frac{\pi}{2}. \end{cases}$$

As long as no edge $e \in E_\nu$ of a net $\nu \in N$ is cut, the cost for the net to zero. If one or more edges cross the cut line, then if all edges follow their preferred directions, the cost is one else the cost is two. This cost function can be easily incorporated into the Fiduccia-Mattheyses heuristic. The only modification necessary is the update of vectors α_ν after a module is moved across the cut line. The update is necessary only for nets connected to the module which is moved. Table 4.1 describes the update procedure for all possible cases.

Every update operation can be executed in constant time. Therefore, the complexity of one pass in the bi-partitioning heuristic remains linear in the total number of edges of the DAG.

4.5 Experimental Results

The proposed bi-partitioning algorithm has been implemented in C language and has been integrated into the GORDIAN placement package [Kleinhans 90]. The proposed method

Module μ moved from	$\mu = \mu_\nu^+$	$\mu \in M_\nu^- \wedge \mu_\nu^+ \in A$	$\mu \in M_\nu^- \wedge \mu_\nu^+ \in B$
A $\rightarrow$ B	$bb_\nu = ab_\nu$ $ba_\nu = aa_\nu$ $aa_\nu = ab_\nu = 0$	$aa_\nu = aa_\nu - 1$ $ab_\nu = ab_\nu + 1$	$ba_\nu = ba_\nu - 1$ $bb_\nu = bb_\nu + 1$
B $\rightarrow$ A	$aa_\nu = ba_\nu$ $ab_\nu = bb_\nu$ $bb_\nu = ba_\nu = 0$	$ab_\nu = ab_\nu - 1$ $aa_\nu = aa_\nu + 1$	$ba_\nu = ba_\nu + 1$ $bb_\nu = bb_\nu - 1$

Table 4.1: Update of α_ν after moving a module $\mu \in M_\nu$

was applied to a few MCNC logic benchmarks [MCNC 88] which were optimized by MIS-II [Brayton 87b]. Table 4.2 shows characteristics of these benchmark circuits.

	circuit 1	circuit 2	circuit 3
modules	552	1636	1657
nets	562	1658	1717

Table 4.2: Examples

The flow based approach proposed here was compared to the standard Fiduccia-Mattheyses method. Cpu-times for calculation of the three placement solutions were 15, 60 and 66 seconds on a DEC3100 workstation for both bi-partitioning approaches. The maximal delays from all primary inputs to all primary outputs (after final placement) were calculated by a standard block-oriented delay calculation technique. The wiring length after final placement was estimated as the sum of the length of minimum rectilinear spanning trees for all nets. As shown in Table 4.3, the flow-oriented approach led to considerable improvements in the timing behavior of the circuit after final placement.

	circuit 1	circuit 2	circuit 3
Fiduccia	1.014×10^3	1.51×10^4	3.23×10^4
Ψ	0.881×10^3	1.24×10^4	3.01×10^4
improvement	15 %	22 %	7 %

Table 4.3: Delay in arbitrary timing units

The comparison in Table 4.4 shows that the improved timing behavior is achieved at the expense of a very moderate increase in the wiring length.

	circuit 1	circuit 2	circuit 3
Fiduccia	9.26×10^4	4.29×10^5	4.92×10^5
Ψ	9.37×10^4	4.59×10^5	5.01×10^5
increase	2 %	7 %	2 %

Table 4.4: Wire length in μ meters

Chapter 5

Logic Optimization

5.1 Logic Decomposition/Restructuring

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. Logic synthesis is often divided into a technology-independent and a technology-dependent phase. In the first phase, transformations are applied on a Boolean network to find a representation with the least number of literals in the factored form. Additional timing optimization transformations are applied on this minimal area network to improve the 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.

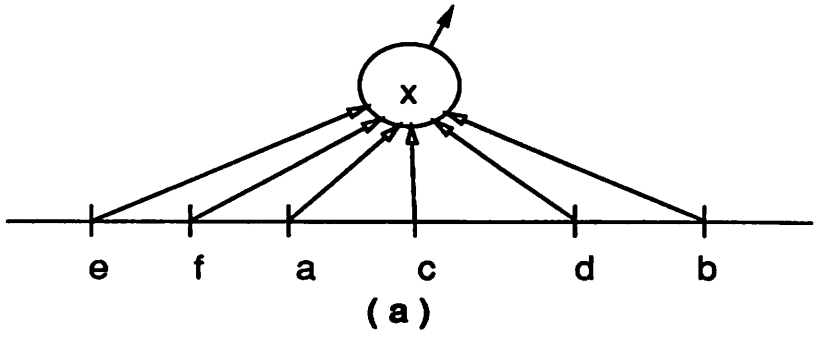
5.1.1 Introduction

Excessive factorization based on common kernel extraction during the technology independent phase of logic synthesis has favored gates with high fanout count and increased path delay. Inordinate attention to minimizing the active cell area has tended toward gates with high fanin count which often increase routing congestion during the final layout and increase interconnection lengths. Attempts have been made to alleviate these problems. Abouzeid

et al. [Abouzeid 90] describe an algebraic decomposition procedure based on lexicographic expressions of Boolean functions in order to reduce gate and wiring areas. This approach aims at expressing the set of logic functions in a form which after technology mapping leads to layered structured cones with ordered input injection. Murgai et al. [Murgai 90] propose a kernel extraction procedure for Table Look-Up PLD's whose objective is to minimize the number of wires created as a result of replacing a node in the network by one of its kernels and the corresponding residue. Approach presented here is different from these approaches, in that placement and logic synthesis are integrated and that wiring cost is estimated explicitly using placement information.

The major issue in decomposition is the identification of common subexpressions. Sharing of such expressions across the design reduces the complexity of the synthesized network. Brayton et al. [Brayton 82] proposed the notion of kernels in algebraic expressions and showed how to use kernels to find multiple cube factors which are common to two or more expressions. Their algorithm selects a kernel which produces the biggest cost reduction in terms of the number of literals. This kernel selection policy tends to minimize the active gate area after technology mapping. Here, I present a decomposition procedure with the objective of minimizing interconnects in the synthesized network.

I shall motivate incorporating the wiring estimates into the kernel selection phase with the following example. Assume that the Boolean network has been placed and that node x and its fanin nodes have positions as shown in Figure 5.1a. The logic expression for node x is equal to $b'c'd' + acd' + ace' + de'f + ce'f + bc'$. Figure 5.1b shows all the kernels and co-kernels for this expression. The value of kernel $e' + d'$ (that is, the number of literals saved if this kernel is extracted and made it into a new node) is one. Similarly, the value of kernel $c + d$ is one. Figure 5.1c shows the final decomposition when $e' + d'$ is extracted while Figure 5.1d shows the final decomposition when $c + d$ is extracted. Both decompositions result in 16 literals (one less than the original 17 literals). However, the interconnect length for 5.1c is more than that of 5.1d. The reason is that the input signals for kernel $e' + d'$ are coming from sources on the placement plane which are placed far apart while those for kernel $c + d$ are coming from sources which are placed near one another. Therefore, the placement information can be used to guide the decomposition procedure in order to minimize the interconnect (often at the expense of little or no cost in terms of the literal

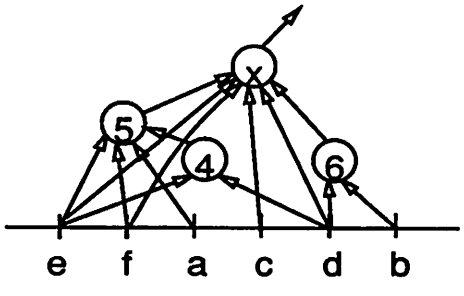


(a)

x = b' c' d' + a c d' + a c e' + d e' f + c e' f + b c'

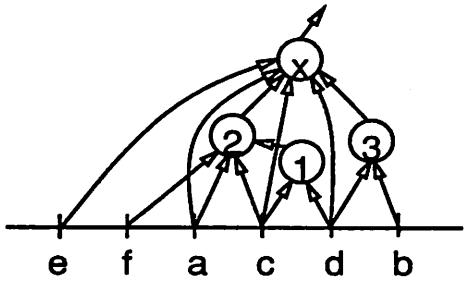
co–kernel	kernel
e' f	c + d
c e'	a + f
a c	e' + d'
c	a d' + a e' + e' f
c'	b' d' + b
d'	b' c' + a c
e'	a c + d f + c f

(b)



[4] = e' + d'
[5] = f e' + a [4]
[6] = b + d' b'
x = c [5] + e' f d + c' [6]

(c)



[1] = c + d
[2] = a c + f [1]
[3] = b + d' b'
x = e' [2] + a c d' + c' [3]

(d)

Figure 5.1: Layout-driven kernel extraction

count.)

After global kernel extraction and node decomposition, the optimized Boolean network must be transformed into a subject DAG consisting of two-input NAND and inverter gates. This process is commonly known as technology decomposition. Here again, placement information can be used to guide the decomposition.

Figure 5.2a shows node y and its fanin and fanout nodes in the layout solution. Assume that $y = a' + b' + c' + d' + e'$. Figures 5.2b and c show two possible decompositions of the five-input NAND function into a DAG consisting of two-input NAND and inverters. For given positions of the fanin nodes, decomposition pattern shown in Figure 5.2c is more desirable than that in Figure 5.2b since there exists an assignment of input signals to pins of the pattern which will allow breaking the five-input NAND gate into two smaller gates (one four-input AND gate and one two-input NAND gate), thereby, reducing the required interconnections.

After choosing the pattern shown in Figure 5.2c, signals must be assigned to pins of the pattern as shown in Figures 5.2d and e. The assignment in Figure 5.2d is undesirable because the decomposition tree conflicts with the placement solution. If this assignment is used, the layout-driven technology mapper will lose the option of reducing the wiring by breaking one big match into smaller matches at the expense of a small increase in the active gate area. Note that this distinction does not arise if the mapper chooses to ignore the wiring cost and minimize only the gate area. Generalizing this observation, a decomposition into primitive gates of the logic function associated with each node in the optimized Boolean network, is sought such that the fanin signals which are coming from nearby regions in the companion placement solution enter the decomposition tree at topologically near points.

5.1.2 Integrating Interconnect Optimization with Logic Restructuring

In this section, techniques for multiple-cube common factor extraction (which subsumes the node decomposition problem) and elimination targeted toward minimizing the total interconnection length of the synthesized network are discussed. These techniques can be combined with those targeted toward minimizing the total number of literals in the factored form representation of the network in order to minimize the routing and active cell area

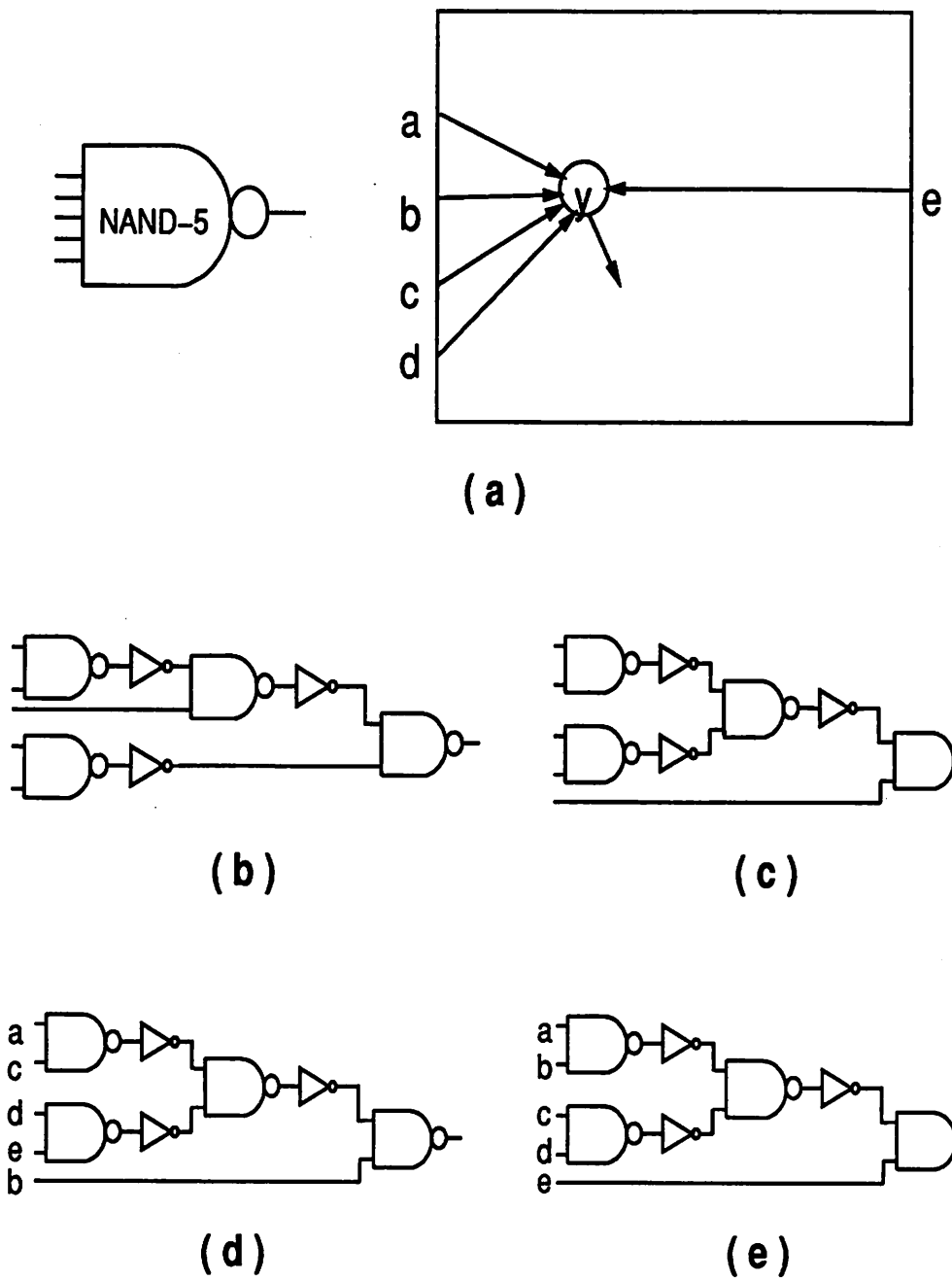


Figure 5.2: Initial choice of subject graph

simultaneously.

Kernel Extraction

The extraction algorithm follows that presented in [Brayton 87a, Brayton 87b]. I concentrate on selection and stopping criteria for layout-driven extraction. In particular, I shall describe the kernel extraction algorithm in detail since the cube extraction algorithm is simpler and follows a similar technique.

The area value of a candidate kernel is considered first. In order to find useful intersections of kernels (which correspond to common multiple-cube divisors between two or more expressions), it is beneficial to construct the co-kernel kernel-cube matrix as in [Rudell 89]. A row in this matrix corresponds to a kernel (and its associated co-kernel), and a column corresponds to cubes which are present in some kernel. The entry B_{ij} is non-zero if kernel i contains kernel-cube j. The product of the co-kernel for a row and the kernel-cube for a column yields a cube of some expression. For reference, the cubes of the original expressions are numbered from 1 to N. The number of the cube resulting from the product of the co-kernel for row i and kernel-cube for column j is placed at position B_{ij} in the co-kernel kernel-cube matrix. A rectangle of this matrix identifies an intersection of kernels; this kernel-intersection is a common subexpression in the network.

The area value of a rectangle (R, C) – denoted by $v_a(R, C)$ – measures the difference in the number of literals in the network if that rectangle is extracted and made into a new node. The number of literals after the rectangle is selected is given by

$$w(R, C) = \sum_{i \in R} w_i^r + \sum_{j \in C} w_j^c$$

where w_i^r is one plus the number of literals in the co-kernel for row i and w_j^c is the number of literals in the kernel-cube for column j. The number of literals before extracting the rectangle is $A(R, C) = \sum_{i \in R, j \in C} V_{ij}$ where V_{ij} is the number of literals in the cube which is covered by position ij of the co-kernel kernel-cube matrix. Then,

$$v_a(R, C) = A(R, C) - w(R, C).$$

The process terminates when $v_a(R, C)$ falls below some user-defined literal saving threshold.

In order to reduce the routing complexity, a kernel-selection policy which chooses a kernel with the greatest cost reduction in terms of the interconnection length is proposed. In particular, the interconnect value of a rectangle – denoted by $v_l(R,C)$ – measures the difference in the total wire length in the network if that rectangle is extracted and made into a new node x. In order to calculate this wire length, node x must be assigned a position and positions of nodes that x was extracted from must be updated. That is, positions of all nodes y : $\exists$ $cokernel(y) \in cokernels(R)$ must be recalculated. An exact solution requires solving a local placement problem which optimally places x and y nodes with respect to their current fanin and fanout nodes in the current network. This local placement problem can be solved efficiently by formulating a quadratic optimization problem with I/O pins located at the boundary of a polygon (and not necessarily a rectangle). (For example, see [Tsay 88, Kleinhans 90, Srinivasan 91].) The exact solution, however, takes more time than is acceptable during the kernel selection phase, and therefore, an approximate solution is calculated: Positions of y nodes are made fixed and x is placed with respect to its fanin nodes and y nodes. Since only one node needs to be placed, the placement update problem is easily solved by placing x at the center-of-mass of its fanin and fanout net enclosing rectangles as will be shown in Subsection 5.2.3.

After assigning a position to x, the new interconnection length is computed as

$$\begin{aligned} v_l(R,C) &= \sum_k (W_{old}(k) - W_{new}(k)) \\ &= \sum_{k \in fanin(x)} W_{old}(k) + \sum_{other\ k} W_{old}(k) - \{W_{new}(x) + \sum_{k \in fanin(x)} W_{new}(k) + \sum_{other\ k} W_{new}(k)\} \\ &= \sum_{k \in fanin(x)} W_{old}(k) - \{W_{new}(x) + \sum_{k \in fanin(x)} W_{new}(k)\}. \end{aligned}$$

where $W_{old}(k)$ ($W_{new}(k)$) denotes the wire length needed to connect node k and its fanout nodes in the old (new) network before (after) extraction. Now,

$$w(R,C) = W_{new}(x) + \sum_{k \in fanin(x)} W_{new}(k)$$

$$L(R,C) = \sum_{k \in fanin(x)} W_{old}(k)$$

and therefore, the interconnect value of a rectangle is given by

$$v_l(R,C) = L(R,C) - w(R,C).$$

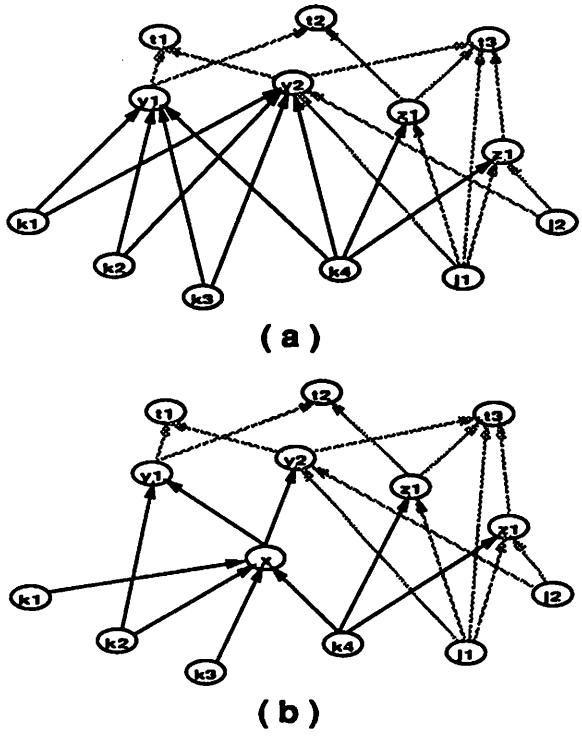


Figure 5.3: Interconnect value of an extracted kernel

Figure 5.3 shows an example of interconnect value computation for a kernel. The dark lines in Figure 5.3a (5.3b) identify the connections which must be considered for calculating $L(R,C)$ $(w(R,C))$. Any of the net models presented in Subsection 5.2.3 can be used in order to compute the interconnection length.

The kernel extraction process terminates when the ratio $v_l(R,C)/L(R,C)$ drops below some user defined wire saving threshold. In order to optimize both literal count and wiring, the value of a kernel is a function of both $v_a(R,C)$ and $v_l(R,C)$.

As new kernels are extracted, the number of nodes and the structure of the network changes. Therefore, the network and its corresponding global placement on layout plane must be updated accordingly. After a new node x is created, positions of x and y's are re-calculated by solving a quadratic optimization problem as described earlier.

Note that interconnect values for overlapping rectangles in the co-kernel kernel-cube matrix are implicitly handled – when kernel x is extracted, its fanin nets are updated and interconnect values of subsequent kernel extractions which overlap x will be calculated based on

this update.

Elimination

The elimination algorithm follows the outline of that presented in [Brayton 87a, Brayton 87b]. Candidate vertices are selected according to some criterion and the elimination takes place if some constraints are satisfied. Elimination terminates when no candidate vertices can be found. I concentrate on the selection and acceptance criteria for layout-driven elimination.

The area value of an elimination is considered first. This value is the difference between the number of literals of the resulting network if the node is eliminated and the number of literals in the current network. This change in the total number of literals in the network (as a result of elimination) is computed by the formula given in [Brayton 87a, Brayton 87b].

The wire length value of an elimination is defined as the difference between the total wire length in the resulting network if the node is eliminated and the wire length in the current network. It is computed in a straight-forward manner from the placement information regarding the node in question and its immediate fanins, immediate fanouts and immediate fanouts of the immediate fanins. Again, in order to optimize both literal count and wiring, the value of an elimination is a function of both area and interconnect values.

5.1.3 Technology Decomposition

The procedure for converting an optimized Boolean network into the subject DAG is not unique and it is an open problem to determine which of the possible subject DAGs yields an optimum solution when an optimum covering algorithm is applied [Brayton 90]. The goal of my technology decomposition procedure is to find a circuit representation which provides a good starting point for the layout-oriented technology mapping. In particular, the logic function associated with nodes in the Boolean network are decomposed such that signals coming from nearby regions of the network enter the decomposition tree at topologically near point(s).

The decomposition process starts by constructing AND-OR trees implementing the sum-

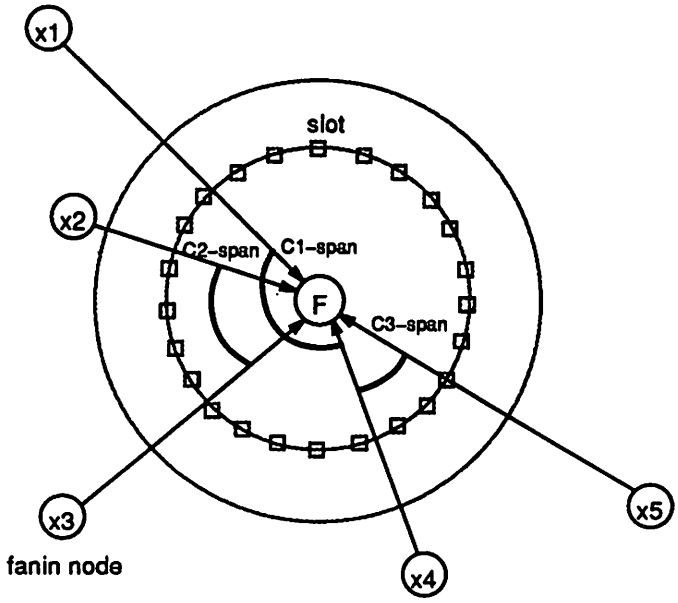


Figure 5.4: Cube ordering viewed as a linear assignment problem

of-product form representation of the logic function associated with each intermediate node in the Boolean network. The function of AND subtrees is to compute the product terms (cubes) and that of the OR subtrees is to compute the sum of the product terms. The input signals to the AND subtrees and then the cubes in the OR subtrees are ordered. The conversion from the ordered AND-OR subtrees to the gates in base function set is straight-forward.

In order to derive the input signal ordering, one refers to the "companion" placement solution for the Boolean network. Each multi-pin net signal is modeled by a star connection from the source toward the sinks. By circularly traversing around each node, a unique ordering is determined for the input signals to the node. This ordering is directly related to the positions of the fanin nodes with respect to the node in question. Next, cube ordering is achieved by setting up a linear assignment problem. S slots are placed on an imaginary inner circle around the node, and the projections of the fanin signals into an imaginary outer circle around the node are found. Then, linear assignment cost matrix $[C]$ is set up whose c_{ik} entry corresponds to the cost of assigning cube i to slot k. This entry is equal to zero if slot k falls inside the shortest circular span for the immediate support of cube i.

Otherwise, the cost is proportional to the angular distance of slot k from the nearest end of the support span of cube i. (See Figure 5.4.)

A linear assignment algorithm [Burkhard 80] is run on the matrix $[C]$. Since rows in the cost matrix $[C]$ correspond to the "floating" cubes and columns correspond to the slots, the linear assignment determines a cube assignment with the minimum sum-cost. The cube ordering is easily derived from the cube positions obtained by the above linear assignment procedure. The process of ordering input signals, cubes and then primitive gate decomposition is recursively applied to all nodes in the Boolean network in order to produce the subject DAG.

5.1.4 Placement Relaxation

After logic synthesis stage, a net list of gates and a companion placement solution are available. The placement solution, however, has overlapping gates and has not yet been mapped to rows (in case of standard cell layout methodology) or to slots (in case of sea-of-gates style). The objective of global relaxation step is to reduce gate overlaps and produce even distribution of gates over the layout image. An additional goal is to make the placement solution feasible. Two basic approaches are generally used for mapping a global placement result to legal locations: (1) Perform a minimum squared error linear assignment which maps the cells in the global placement to the legal positions simultaneously; (2) Use a hierarchical bi-partitioning technique to obtain a feasible placement solution.

I have adopted the top-down bi-partitioning heuristic in the following way. The placement procedure consists of alternating and interacting global optimization and partitioning steps. In particular, for a circuit with M gates, the placement procedure goes through $m = \lceil log_2 M \rceil$ steps in order to produce a detailed placement. Now, assume that an initial placement solution for the circuit and two parameters N_s and N_f are given. These parameters specify the start and finish conditions for the relaxation procedure, that is, relaxation begins when number of modules per hierarchical region is N_s and ends when this number is N_f. Let $s = \lceil log_2 N_s \rceil$, $t = \lceil log_2 N_f \rceil$, then $m \geq s \geq t \geq 0$.

The objectives are 1) to maintain structure of the initial placement solution by skipping earlier global optimization and partitioning steps and 2) to distribute gates evenly over the

circuit boundary by doing global optimization and partitioning at the later steps. The placement procedure is, therefore, modified such that it goes through steps $m-s, \cdots, m-t$ only, thereby, achieving the relaxation goal without drastically disturbing the initial placement solution. Note that the final mapping to rows is performed when $N_f = 1$.

5.2 Technology Mapping

5.2.1 Introduction

Given a Boolean network representing a combinational logic circuit optimized by technology independent synthesis procedures and a target library, technology mapping is the process of binding 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 [Keutzer 87] and MIS [Detjens 87]. 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 [Aho 76]. DAGON and MIS technology mappers generate circuits with small active cell area but ignore area and delay contributed by interconnections between gates.

I justify incorporating wiring estimates into technology mapping by pointing out problems associated with minimizing only active cell area. Figure 5.5.a 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. The 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 gatess_i can be placed near the matching gates. However, in many cases, these gates

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

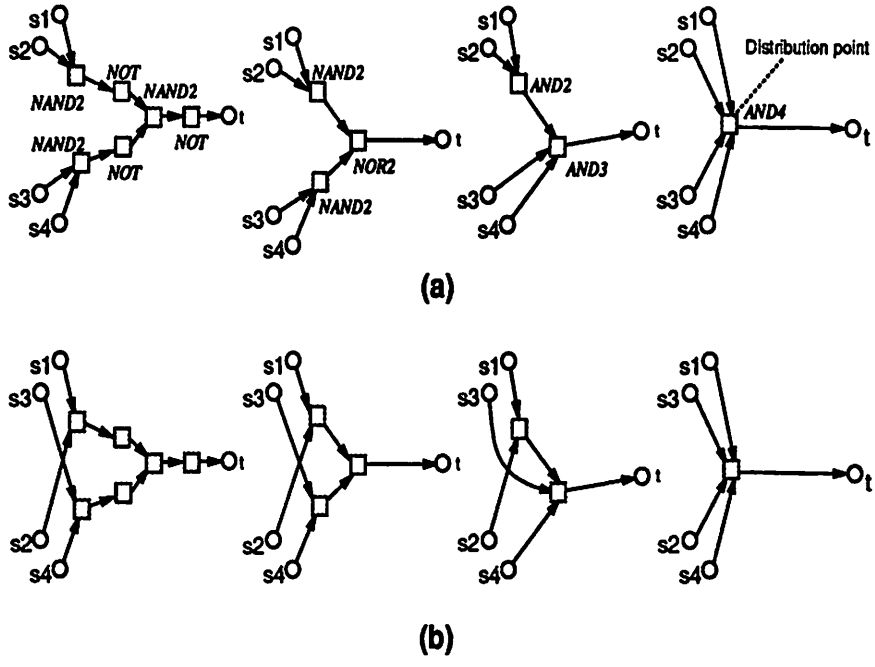


Figure 5.5: Active gate area versus wire length trade-off

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 5.5. 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.

Figure 5.5.b illustrates the importance of a good decomposition for the layout-driven technology mapping scheme. This figure shows the same decomposition tree as in Figure 5.5.a. 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. Therefore, Figure 5.5.a provides a better decomposition (and hence potential for higher quality mapping) than that in Figure 5.5.b.

I present a technology mapping procedure based on DAG covering which integrates gate placement and interconnection length estimation with the dynamic programming algorithm. LILY's mapper 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. The interconnection dependent contributions to circuit area and delay is estimated 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.

5.2.2 Terminology

The DAG covering approach to 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 performance 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 two-input NAND and inverter gates. This is the network in its unmapped form which

will be referred 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.

Consider Figure 5.6 which shows an example $N_{inchoate}$ at some point during the mapping process. Assume that cone K_1 (corresponding to primary output po_1) and some of the nodes in cone K_2 have been mapped. The remaining nodes in cone K_2 as well as nodes in cone K_3 must be mapped next. (In the dynamic programming approach, mapping starts from the primary inputs of a logic cone and nodes are recursively processed 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. It cannot be predicted whether or not a *nestling* will be present in the final mapped network, N_{mapped}, until po_2 is reached. 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 5.7.) At the end of the mapping procedure, only *hawk*s and *dove*s remain. This classification will be used in Subsection 5.2.3 for dynamic position calculation.

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.

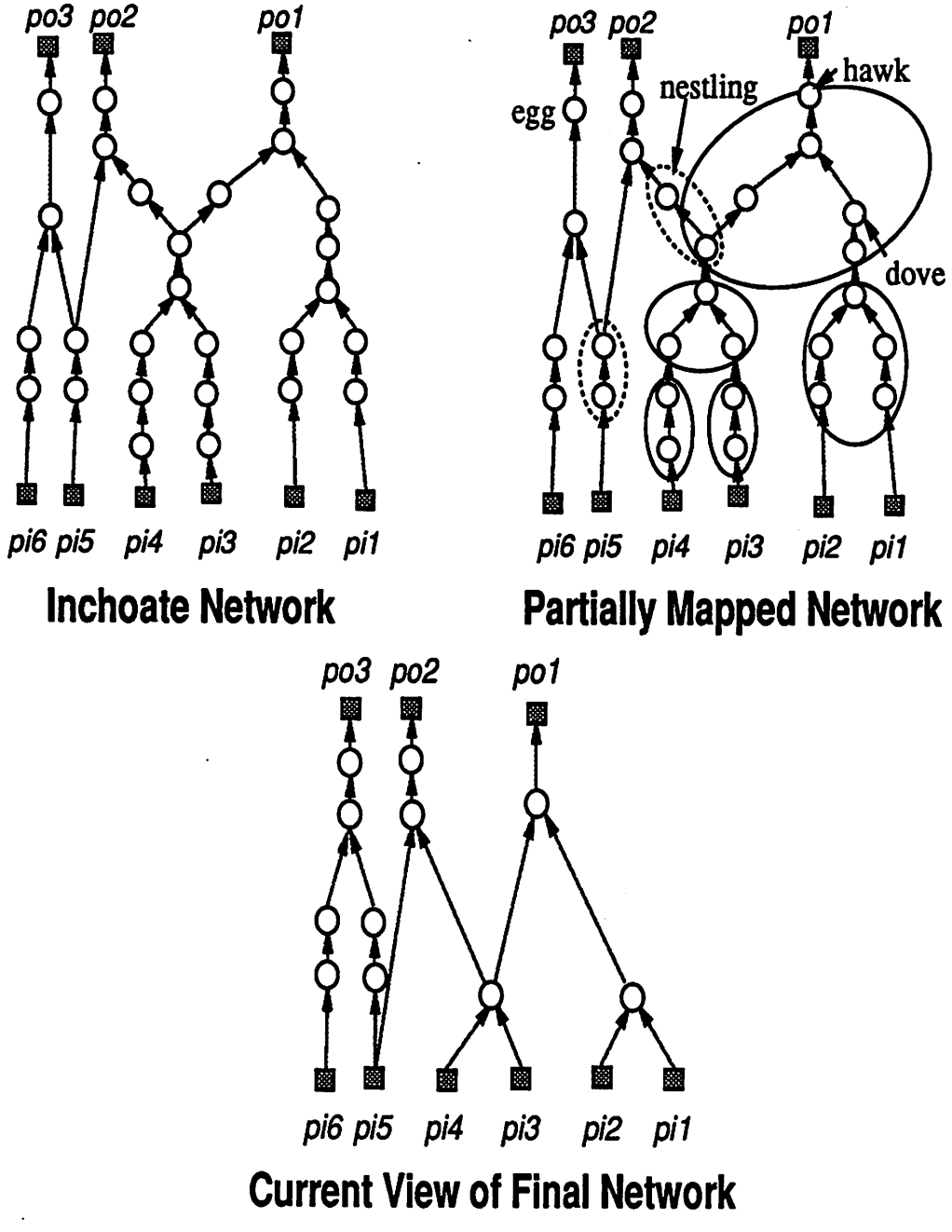


Figure 5.6: Incremental updating of the Boolean network

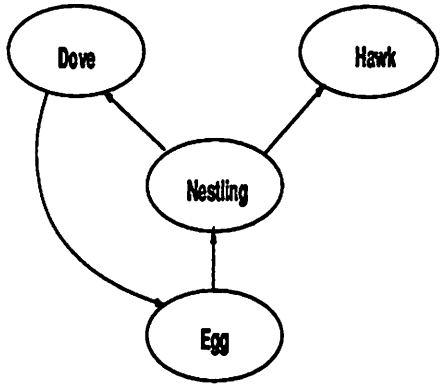


Figure 5.7: A node's life cycle during the mapping

5.2.3 Technology Mapping for Minimum Layout Area

The goal is 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 the cost of match m at node v is to calculated. (See Figure 5.8.) This cost consists of two components:

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

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

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 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 $wire\_cost(v,m)$ 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 $gate(v_i)$.

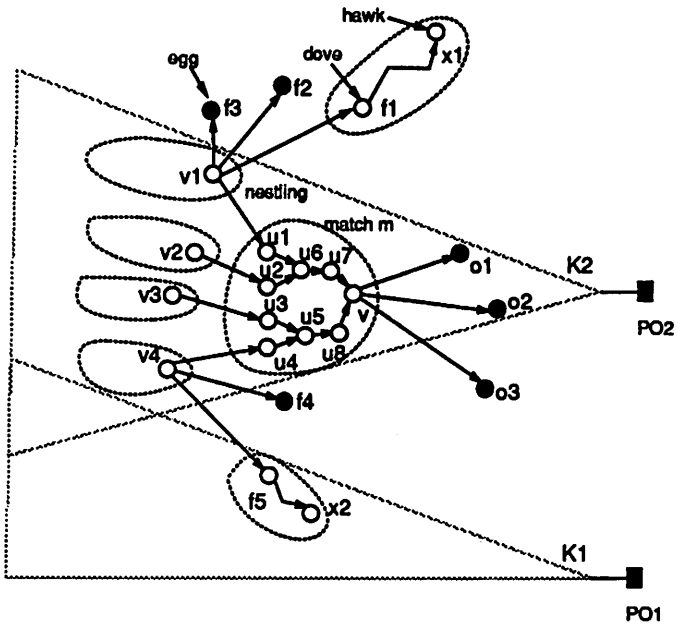


Figure 5.8: Cost calculation for a candidate match

Initial Placement of the Network

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. By a balanced global placement, it is meant that 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. The global optimality of the solution and the ability to capture the logic structure are endangered if gates are prematurely forced into rows or slots. Secondly, the placement updating procedure does not perform well on a fixed two-dimensional mesh due to the inability to keep track of the slot densities during the dynamic programming step.

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, which coincide with the point placement model, do not introduce much

error when the number of gates in the circuit is large.

Finding Dynamic Fanouts

The *dynamic fanouts* of fanin v_i for match m at v (which is a node in cone K_i) are found in the following manner. (See Figure 5.8.) Dynamic fanout refers to a fanout of v_i that is present at the current step. A dynamic fanout is a *hawk*, a *nestling* or an *egg* which has v_i as its fanin. For example, the list of dynamic fanouts of node v_1 consists of nodes v, x_1, f_2 and f_3. Due to logic duplication, it is possible to find more than one dynamic fanout along a given branch. Fanouts of m are the same as fanouts of the root of the match in the inchoate network and are calculated statically.

During dynamic position calculation, *mapPositions* are used for *hawks* and *placePositions* are used for all other node types.

Dynamic Updating of Placement

Initially, nodes in $N_{inchoate}$ are assigned valid *placePositions* based on the global placement solution. As nodes are mapped, *mapPositions* are calculated and stored on nodes. There are three options for computing the *mapPositions*. In the *CM-of-Merged-Nodes* option, m is placed 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 5.8.) In the *CM-of-Fan-Rects* option, m is placed at the center of mass (or median) of its fanin and fanout rectangles. In the *CM-of-Fan-Nets* option, m is placed at the center of mass (or median) of its fanin and fanout nets. Due to depth first search ordering used during the mapping procedure, $inputs(v, m)$ have already been mapped and therefore their *mapPositions* are used; $outputs(v)$ are not mapped yet and their *placePositions* are used.

The advantage of *CM-of-Merged-Nodes* option is that the *mapPositions* are always calculated by referring to the initial global placement solution. Since the initial placement is balanced and captures the adjacency relations and directions of signal flow between nodes in $N_{inchoate}$, the dynamically evolving placement will also be balanced. The disadvantage

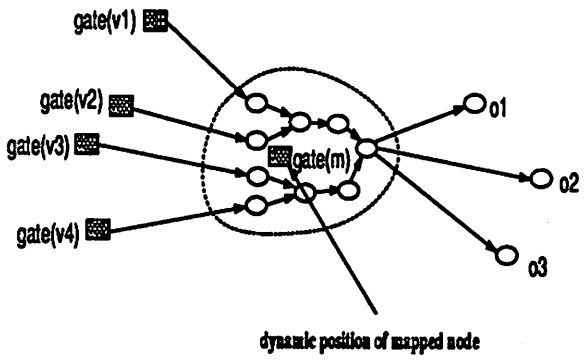


Figure 5.9: Dynamic updating of placement positions using *CM-of-Merged-Nodes* option (Euclidean norm)

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 policy is often pessimistic. (See Figure 5.9.)

The remaining options are, in essence, techniques for *constructive* placement of the network being mapped. Note that because of dynamic programming formulation, as many constructive placement solutions as there are enumerated mapping solutions are generated and stored. A mapping solution along with its associated placement solution are finalized after mapping the root node of each logic cone.

The advantage of *CM-of-Fan-Rects* option is that m is placed at a position which causes minimum increase in the wire length with respect to its "fixed" fanin and fanout which is desirable. The disadvantages are that the placement may become unbalanced and 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. The first difficulty 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, *eggs* and *hawks* are assigned *placePositions* based on the new placement result. The second difficulty is more subtle. One solution is to perform a preprocessing pass on the network during which for each node v, all possible *outputs*(v) are recorded. This is accomplished by finding every possible match in the network which has v as an input. During this preprocessing phase, matches are placed at the center of mass of their merged nodes. Clearly,

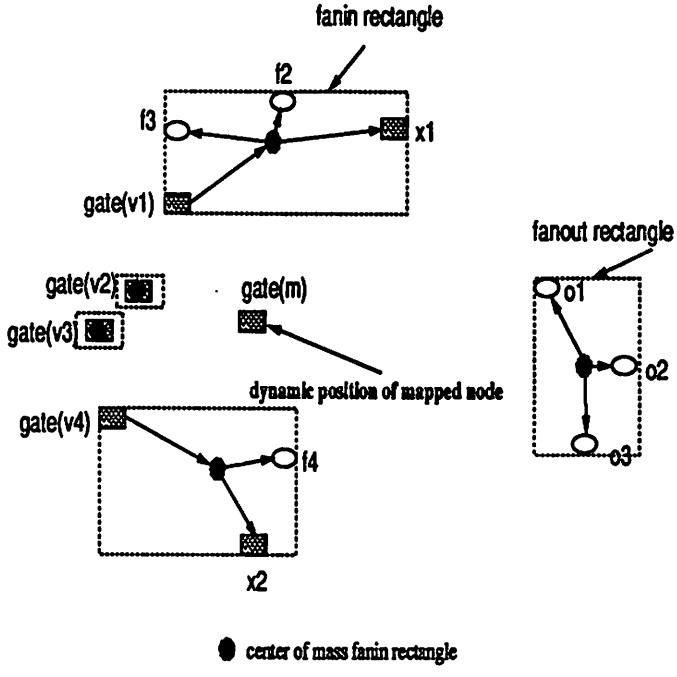


Figure 5.10: Dynamic updating of placement positions using *CM-of-Fan-Rects* option (Euclidean norm)

this technique leads to a slow-down of LILY since all different matches at the $outputs(v)$ must be considered before choosing match m at v.

When using *CM-of-Fan-Rects* option, depending on the wire length metric adopted, the problem can be solved efficiently or can become difficult. Consider Figure 5.10, 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. That is, 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

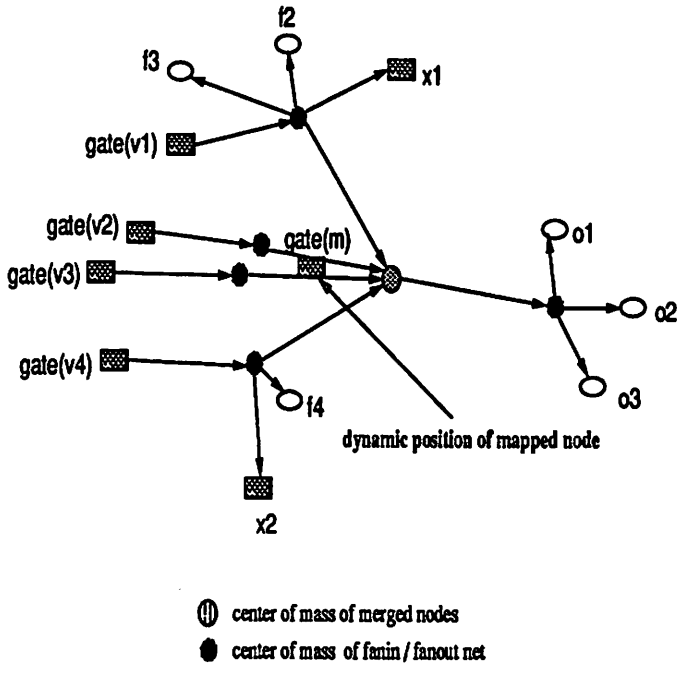


Figure 5.11: Dynamic updating of placement positions using *CM-of-Fan-Nets* option (Euclidean norm)

each of the rectangles. The problem is a special case of solving for the median of a graph which is presented in [Hakimi 64]. 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.

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, each fanin/fanout rectangle is represented by its center of mass point, then the optimal point location problem is solved by computing the center of mass of these points. Note that when constructing the fanin/fanout rectangles, nodes of $merged(v,m)$ are excluded from the fanin/fanout nets.

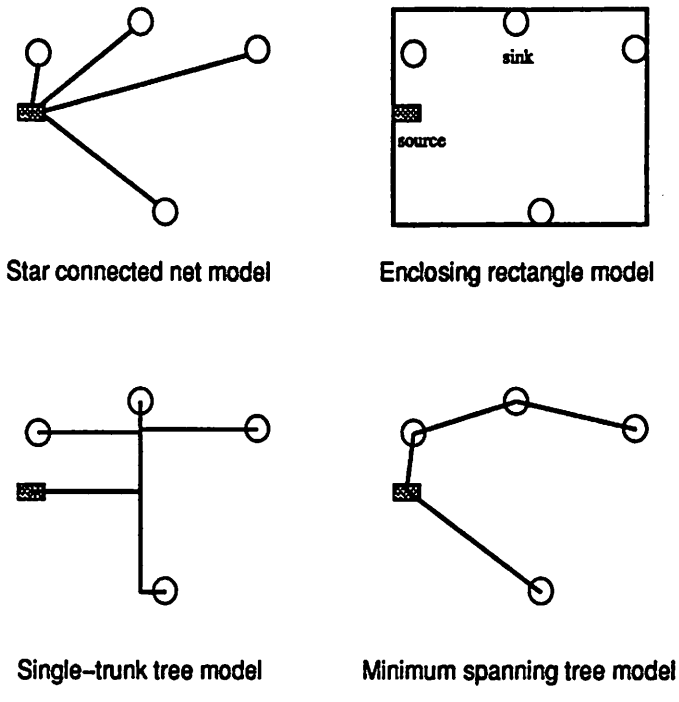


Figure 5.12: Various connection models for multiple pin nets

CM-of-Fan-Nets option is the combination of the first two options. Here, first a position for m is calculated using *CM-of-Merged-Nodes* option, then the center of mass (or median) for all the nets which dynamically connect to m are calculated. Finally, m is placed at the center of mass (or median) of the center of masses (or medians) for these nets. As a result, sensitivity of the dynamic placement update procedure on the exact locations of the dynamic fanin and fanout nodes is reduced, the circuit is remains balanced and the wire cost increase is kept small. (See Figure 5.11.)

Wire Cost Estimation

After positioning $gate(m)$, the wire cost associated with the matching of m at node v must be calculated. This cost consists of the sum of the wire lengths from m to its fanins. Consider fanin v_i of m. If it is driving only the input pin of m, the wire length calculation is simple. However, if it is driving multiple fanout pins (including input pin of m, of course), then the calculation becomes more involved.

The following approximate procedures for calculating length of a multiple pin net have been implemented (Figure 5.12)

- **Star connected model** — Assume a direct source to sink connection pattern and therefore calculate distance from the source pin to each sink pin and sum over;
- **Enclosing rectangle model** — Calculate the half perimeter length of the minimum box bounding all pins on the net;
- **Single-trunk tree model** — Assume that pins on the net (sinks and the source) connect to a single trunk that passes through the center of mass (or median) of all pins either in horizontal or vertical direction. The direction leading to minimum trunk length is used;
- **Minimum spanning tree model** — Find a minimum spanning tree connecting all pins on the net and calculate its length.

Note that during technology mapping for minimum area, only length of the line connecting each fanin v_i to m is of interest. Therefore, the net length calculated using the above models must be divided by the dynamic fanout count at v_i in order to get the expected wire length contributed by connection from v_i to m and thereby avoid duplicate accounting of the wire cost. In fact, using the star connection model, the edge length from v_i to m can be directly calculated. Other models provide an average edge length.

Cone Ordering

The dynamic fanouts corresponding to the *hawk*s will necessarily exist in the final network. Other dynamic fanouts are tentative, in the sense that they may not exist in the final network. However, all dynamic fanouts are needed for calculating the wire cost of a match as described above. Therefore, an ordering of output cones that minimizes the number of references to the dynamic fanouts which have not been mapped yet is attractive. Reconvergent stem nodes whose reconvergence region is a subset of exactly one logic cone give rise to *egg* or *nestling* dynamic fanouts inside the current logic cone. However, this situation cannot be avoided. Therefore, an ordering which only minimizes the number of references

to the *eggs* outside the current logic cone is pursued. 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}. An $n \times n$ matrix M is constructed such that its ij entry holds $E(K_i, K_j)$. 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.

5.2.4 Technology Mapping for Minimum Circuit 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. Hence, it is only natural that wiring delay is incorporated into the calculation of the arrival time.

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. Therefore, the intrinsic delay from input i to y is denoted by I_i, and the output resistance at y corresponding to input i is denoted by R_i. I_i and R_i have separate values each for rising and falling delays.

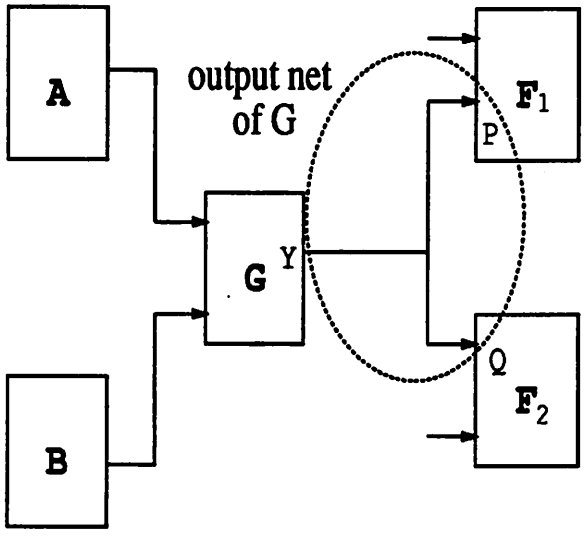


Figure 5.13: Output load consists of fanout load and wiring load

Based on this model, the arrival time [Burstein 85] 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, the output arrival time can be written recursively 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.

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. (See Figure 5.13 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 the interconnections have been modeled by a lumped capacitance, $t_y = t_q$, i.e., the 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, 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 connection models described previously. For loading calculation, the length of the net (and not that of the edge connecting source to sink) should be used.

Updating the Arrival Time

During the mapping process, when m is matching at node v, 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 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 [Heinbuch 88]). However, in order to calculate the wiring capacitance, positions of gates at the node's fanout must be known. This is not possible and instead positions of these gates are read from the initial placement solution of $N_{inchoate}$. This simplification gives rise to inaccuracies in the arrival time calculation.

To prevent the inaccuracy from propagating through, the following observation is used: When matching m at v, the capacitance at the output of *inputs(v,m)* is known because the type and position of their fanout gate which is *gate(m)* is known. If the output arrival times of *inputs(v,m)* are updated, then the input arrival time of *gate(m)* is accurate. The splitting of the arrival time calculation into load independent 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. (See Figure 5.14.)

Corresponding to each input i, the *block arrival* time at g is defined 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

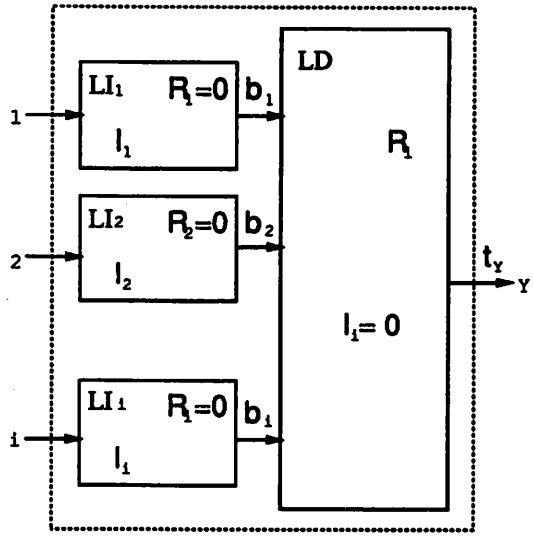


Figure 5.14: Gate splitting for timing re-calculation

$t_y = max\{b_i + R_i\ C_L\}$. The advantage of this splitting is that only the $R_i C_L$ part has to be recalculated for different loads - b_i's remain unchanged.

Mapping for Minimum Delay

Consider the mapping scenario at node v in Figure 5.8. Block arrival times at v_i have already been calculated. 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. The list of dynamic fanout nodes for v_i are found, and match m is added to this list. The current load, seen at v_i, is calculated from this list. The input capacitance of $gate(m)$ and its *mapPosition* are used for calculating C_L and C_w respectively. For a *hawk* in the list, the input capacitance and the *mapPosition* for $gate(hawk)$ are used. For an *egg* or *nestling* in the list, the input capacitance and the *placePosition* for its base function gate are used.

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.

5.3 Fanout Optimization

5.3.1 Introduction

Fanout optimization is an important step in logic synthesis. It can significantly improve the circuit performance by reducing the delays. It can also be used to enforce fanout constraints or load constraints imposed by a technology. The delay through a gate is approximately proportional to the number of fanout sinks. Hence, restricting the fanout at gates, the delay can be reduced. (The delay can also be reduced by transistor sizing and improved drive capability of the gate but these issues will not be pursued here.) Commonly used techniques for fanout optimization include buffer insertion and logic duplication. Both techniques often achieve reduced delays at the expense of increased area.

Most logic synthesis systems use buffer trees after technology mapping. It is, however, desirable to move the fanout optimization process to higher levels, i.e., to canonical decomposition or technology mapping steps. One way of controlling the fanout during technology mapping is to build buffer trees for the inchoate network. A timing analysis on the inchoate network identifies the high-count multiple fanout points (fanout > fanout-limit) which lie on the critical paths. The high fanout-count at these points is likely to be preserved during mapping, and therefore, buffer insertion will be required after the mapping. By inserting buffer trees at these critical high-fanout points, the possibility of high-count fanout points after mapping is reduced and more realistic area and delay estimations are obtained during the mapping process.

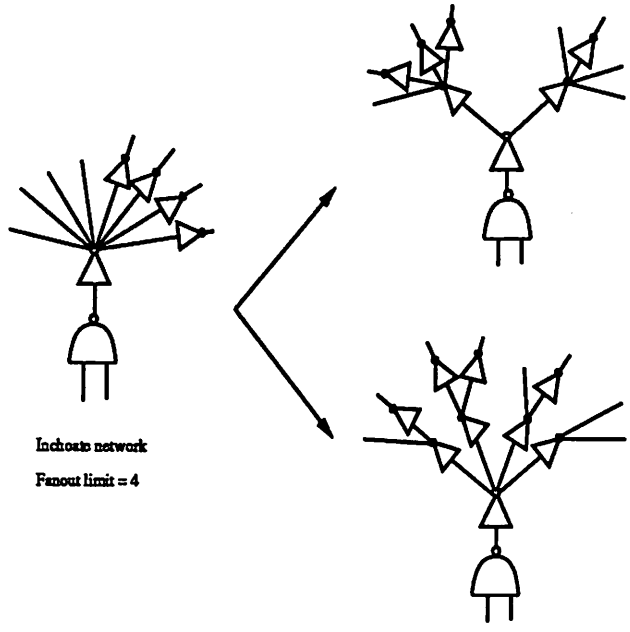


Figure 5.15: Buffer insertion

5.3.2 Buffer Insertion

The buffer trees are constructed as follows. Let m be the number of sinks and k be the fanout limit to be enforced. The source then drives k buffers, each of which in turn drives k gates or buffers. The number of levels required is $log_k m$. For most practical purposes one level of buffers is sufficient. m/k buffers are put at the level 1 and the sinks are divided equally among them.

There are two choices for buffer trees as shown in Figure 5.15. The first tree has more buffers at level 1 each of which is driving a smaller number of sinks. The advantage of the first tree is that it has more freedom during placement due to the reduced forward cone dependencies but the second tree has lower area, lower routing complexity (lower number of nets), and provides more critical signal isolation. Thus, I choose the second form of tree.

The methods described in [Touati 90] of building fanout trees based on the load and timing information at the sinks cannot be reliably used at this step due to lack of accurate load information. However, sinks can be divided among buffers to reduce the routing complexity by driving sinks which are going towards the same outputs by the same buffer. This is

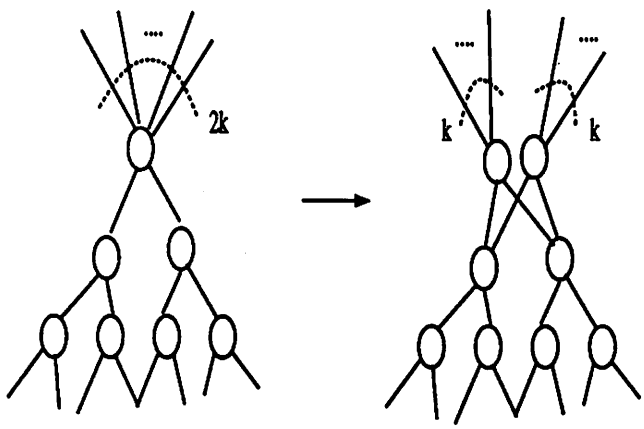


Figure 5.16: Changes to inchoate network for logic duplication

done by creating a block for each fanout and creating pins on the block corresponding to the primary outputs in the transitive fanout cone of the sink. Then, a linear placement of the blocks which minimizes the total net span is obtained. The ordering of the sinks so produced maximizes the proximity between the sinks. It has been observed that this method of modifying the inchoate network helps reduce the maximum fanout in the mapped circuit at almost no cost in area.

5.3.3 Logic Duplication

Another technique for fanout optimization is through logic duplication. A gate which is driving a large number of sinks is duplicated and the sinks are divided among them. This method of optimization is more effective in reducing delays compared to the buffer insertion which may introduce extra delay due to the increased levels of logic. (See Figure 5.16.)

Logic duplication can be performed after technology mapping but the gates which need to be duplicated tend to be larger which makes the area cost of duplication high. A better way is to handle duplication at the time of technology mapping or even before the mapping. The required duplication is done in the inchoate network. The advantage is that fanout limits can be satisfied by duplicating potentially smaller gates during the mapping process. The transformed inchoate network allows all the possible solutions generated for the original inchoate network as well as additional solutions which would not be possible with the

original inchoate network.

5.4 Experimental Results

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 along with the 'big' library, a layout-driven mapping procedure is used, then 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 $\widehat{W} < min(W_{tiny}, W_{big})$ will be produced.

The objective was to show that by integrating logic synthesis 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, two pipelines were used to produce the results: 1) Read in the optimized circuit; do technology decomposition; run MIS technology mapper in area or timing mode; write the 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; do placement driven technology decomposition; run LILY in area or timing mode; write the mapped circuit and "companion" placement solution to the database; do routing. In both cases the same placement, pin assignment and routing tools were used. Note that the first option which is the standard MIS pipeline cannot make use of the location of pads during the decomposition or technology mapping process.

The benchmarks were first optimized for minimum area using the *rugged script* [Savoj 91]. This script produces the *area optimized* circuits. Next, the *delay script* [Touati 91] (which does a quick decomposition, resubstitution, depth reduction, redundancy removal and full simplification) was run on the area optimized circuits to produce the *delay optimized* ones.

The literal count results (before technology mapping) are depicted in Tables 5.1.

Ex.	original	area opt	delay opt
9symml	277	183	200
C1355	1032	552	832
C1908	1497	535	841
C3540*	2934	1283	1629
C432	372	219	317
C5315	4369	1763	2494
C6288*	4800	3367	4231
C880	703	414	558
apex6	904	732	1002
apex7	289	243	334
b9	236	124	153
rot	764	664	797

Table 5.1: Multi-level benchmarks: number of literals in factored form (* means full_simplify was not used)

Table 5.2 shows comparisons between LILY and MIS2.1 results in terms of active cell area, total chip area and total interconnection length (for area optimized circuits and mapping in area mode). In general, LILY's mapper tends to use smaller gates, larger active cell area (avg. 1%) but smaller total chip area (avg. 3%) and interconnection length (avg. 3%).

example	MIS2.1			LILY		
	inst. area mm^2	chip area mm^2	wire length mm	inst. area mm^2	chip area mm^2	wire length mm
C1355	0.458	1.311	136.7	0.454	1.257	129.7
C1908	0.515	1.720	192.1	0.520	1.665	188.3
C3540	1.384	5.737	694.6	1.391	5.588	672.1
C432	0.246	0.742	84.2	0.258	0.696	77.8
C5315	1.700	7.694	920.9	1.721	7.611	903.9
C880	0.452	1.458	158.8	0.453	1.357	148.3
apex6	0.728	3.194	386.7	0.737	3.010	378.5
apex7	0.258	0.720	81.1	0.265	0.817	89.1
b9	0.147	0.350	35.3	0.149	0.344	35.1
rot	0.747	3.035	376.6	0.759	2.929	369.8

Table 5.2: Comparison of the total instance area, final chip area and interconnection length after detailed routing

Table 5.3 shows comparisons between LILY and MIS2.1 results in terms of total chip area

and longest path delay (for delay optimized circuits and mapping in timing mode). The delays are based on a 1μ standard cell library. (Since information on a real 1μ library was not available, the delay, gate capacitance and wiring capacitance of 3μ technology were scaled appropriately [Heinbuch 88].) 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 area improvement of 27% and delay improvement of 12% compared to MIS2.1.

example	MIS2.1		*LILY*	
	chip area mm^2	delay ns	chip area mm^2	delay ns
9symml	1.004	9.02	1.182	9.53
C1355	6.231	18.09	4.100	15.15
C1908	7.923	32.60	6.774	24.98
C3540	21.325	40.11	13.020	36.00
C432	3.215	24.00	3.051	20.81
C5315	28.331	28.04	17.45	25.06
C880	3.226	20.67	3.158	19.68
apex6	6.160	15.91	5.818	13.60
apex7	1.376	8.27	1.324	8.50
rot	5.803	15.78	5.921	15.34

Table 5.3: Comparison of the final chip area and longest path delay results after detailed routing

The following tools and options were used: GORDIAN [Kleinhans 90] package for global placement, *CM-of-Fan-Nets* option for dynamic placement update, Euclidean distance metric, enclosing rectangle connection model for wire length estimation, PACT pad placement program, TimberWolf 4.2 [Sechen 85] global router, and YACR [Reed 85] 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.

It is 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.

LILY's 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 [Touati 90], one could perform a preprocessing pass during which for each node, all possible load values are recorded (by examining every possible match) or could perform a postprocessing pass to derive fanout trees.

Part II

Physical Design

Chapter 6

System Overview

6.1 Motivation

Macro-cell layout style is often used when a complex circuit must have minimum area and high performance. This layout style has advantages in terms of both area and performance over other cell-based layout styles with regular structures (e. g., standard cell and gate array/sea-of-gates). The price to pay is higher design cost and longer design time. The macro-cell design aid tools are not as mature as standard cell or gate array tools, and hence, macro-cell layout requires more time and effort on part of the chip designers. The macro-cell layout, however, uses more flexible topologies and cells, hence allowing very good area and performance optimizations. In addition, this layout style is more suited to a hierarchical design environment since it relies mainly on interface description of the cells, and therefore, permits higher degrees of design modularity and concurrent group work. The cells to be laid out in a macro-cell assembly can each be optimized independently based on the functionality and requirements of the cell. Individual cells can be implemented by the most appropriate layout style.

6.2 BEAR-FP: A Macro-Cell Layout System

BEAR-FP [Pedram 90c] is a macro-cell based layout system which builds on its predecessor, the BEAR system [Dai 87d]. It inherits BEAR's dynamic and efficient data representation, which unifies topological and geometrical information, and BEAR's routing system. BEAR-FP, however, has a new floorplanning procedure with integrated global routing and hierarchical pin assignment, a new Steiner-tree global router, a detailed channel pin arrangement procedure, and a timing-driven clustering and placement capability. BEAR-FP supports traditional macro-cell layout with routing channels as well as channel-free layout style.

The guiding principle of BEAR-FP is that of stepwise refinement, which means that the geometrical positions and interface characteristics (shape and pin locations) of cells are determined gradually and in a top-down fashion. For example, positions of floating pins on cells are initially determined based on minimizing total wire length in the chip. These positions are later modified to honor channel capacity constraints at a minimal increase in wire length. Before detailed routing, pin positions within a channel can be changed to minimize the routing density. As another example, shapes of flexible cells are determined during the top-down floorplanning. These shapes can be further optimized after global routing (when more detailed information about connection paths exist).

BEAR-FP provides a complete pipeline from a net list specification of a circuit to a finished layout. The user specifies various physical, timing and topological constraints, controls the order that various optimization tools are invoked and sets parameters to control the outcomes of these procedures. BEAR-FP addresses the issue of performance optimization by including a scheme for timing-driven layout that spans the entire layout process, i.e., the timing constraints influence the clustering, floorplanning, pin assignment and global routing steps.

BEAR-FP can be used for "early" floorplanning as well. In this mode, it does a quick floorplanning whose results can be used to guide the logic synthesis procedure by giving the logic design tools information about the interface characteristics (shape and pin distribution) of the blocks and about how the physical design process can impact the timing on certain critical paths of the circuit. In the following paragraphs, a typical design flow is examined and in the process the set of tools which constitute BEAR-FP are briefly described.

Initially, a hierarchical representation of the circuit in the form of a multi-way cluster tree is generated. Each leaf in the tree corresponds to an actual cell and each internal node (which is called a *cluster node*) represents a collection of highly connected cells (or clusters of cells). This tree is generated bottom-up and is obtained by minimizing connections among various cells. The maximum branching factor in the tree is restricted to a small value (e.g., four). This restriction is necessary because the number of multi-way floorplan patterns increases dramatically as the branching factor is increased.

Top-down floorplaning [Dai 89, Pedram 90a, Pedram 90b] is started from the root of the cluster tree and is continued in a breadth first manner. As a result of floorplanning a cluster node, its child nodes are assigned shapes and positions, and the current partial floorplan solution is updated accordingly. Next step is pin assignment and global routing. An initial pin assignment produces a solution minimizing the total interconnection length. Global routing produces shortest connection paths for all nets. If capacity constraints for some channels are violated, a new pin assignment re-positions the floating pins in order to reduce congestions in the over-subscribed channels. The process of the top down traversal of the cluster tree continues until the leaf level is reached. Because the global pin assignment procedure does not find the optimal pin locations within each routing channel, it is followed by a channel pin arrangement procedure [Cong 89].

The global router [M-Sadowska 86] takes a subregion in which routing has not yet been completed and determines a cut which separates it into two smaller subregions. When a net has pins or pseudo pins in both sides of the partition, the net crosses the cut line. For each such net, pseudo pins are inserted along the cut line in appropriate bottleneck tiles. This cutting process continues until each subregion on the list is free of bottlenecks. At each partitioning step, the linear assignment algorithm is used to determine where the net will be placed. The global router tries to place nets in bottleneck tiles so that the density (used space) does not exceed the capacity (available space) of each tile. The global router works directly on a plane and does not have any knowledge about the division of channel routing regions. Such an approach allows for better area utilization and it sets the scene for block placement corrections and corresponding adjustments of routed nets.

After the global router completes its job, topologies and positions of all nets with respect to blocks are determined. Since it is quite difficult to estimate the routing area precisely,

some bottleneck tiles will have more nets passing through them than their sizes permit. Similarly, some tiles will have less. The purpose of *global spacing* [Dai 87b] is to match the capacity of each bottleneck tile with its density as much as possible while preserving the existing nets' topologies. There are two steps in global spacing. The first step, global decompaction, eliminates negative mismatches (density ¿ capacity) at the expense of a minimal increase in the chip area. The second step, global compaction, reduces the chip area as much as possible without creating negative mismatches. During global spacing, global routing is updated incrementally. A dynamic data representation, which unifies topological and geometrical information, is used to achieve an efficient implementation of such difficult operations [Dai 87a].

The floorplanning procedure described above, chooses the best shape for a flexible cell from a finite set of discrete shapes. The shape optimization procedure, however, assigns a shape to the flexible cell by varying its aspect ratio (while keeping its area fixed) between a minimum and maximum aspect ratio. The pin assignment performed during this procedure is a simple scaling of pins on the sides of cell such that the relative spacings between pins are intact. At this point, the space between any two blocks as well as the space needed for routing is known. The main advantage of this method is that it can use the information provided by the placement *and* the global routing to accurately model the influence of block shape modifications on the overall chip size. After resizing a cell, the global routing information around the cell is incrementally updated.

BEAR-FP allows an additional opportunity to refine the placement during the local routing. To make such refinement robust and efficient, a feasible routing order is crucial. In a feasible routing order, when a new channel is being routed, it can be expanded or contracted without destroying the previously routed channels. Hence, routing completion is facilitated without iteration. Rather than restricting the floor plans or placements to slicing structures (in such special case, finding feasible routing order is trivial) as done in most other system, BEAR-FP provides a feasible routing order for non-slicing structures by introducing L-shaped channels [Dai 85].

The channel router *Glitter* [Chen 86, Chen 87] which is a gridless, variable width router does the detailed routing. Glitter places horizontal wires on one layer and vertical wires on the other layer. The channel compactor *Nutcracker* [Xiong 87] can be used to reduce the

channel height after routing. This strategy may lead to many more vias than necessary. The detailed routing is therefore followed by a *via reducer* [Xiong 88] which slides and removes unnecessary contacts. The last step in detailed routing is *ring router* [Wang 90] which connects the core of a chip to the I/O pads at the periphery. The ring router expects all signal wires to be the same width. Wires specified as power/ground can be of arbitrary widths.

In the following chapters, I will focus on the floorplanning and pin assignment programs within BEAR-FP.

Chapter 7

A Robust Framework for Floorplanning

7.1 Introduction

During the early stages in the design of electronic systems, decisions are made which have a dramatic effect on the quality (performance, density or area) of the resulting design. Choices must be made in partitioning functions into physical cells and in choosing interface characteristics of the cells such as size, shape, and pin positions. These choices are difficult because their effects on the circuit area and speed may not become known until much later. Floorplanning helps solve these problems.

The objective of floorplanning is to trade off cell interface characteristics (size, shape, and pin position) and cell locations to optimize the layout. Floorplanning is the first step in the physical design which determines the spatial and interface characteristics of given cells such that the desired physical and electrical constraints are satisfied. The floorplanner must generate a chip plan that can be implemented by cell generators. Such a floorplanner would allow better automation of the physical design process.

Floorplanning is useful in yet another way. One must know something about how a design is to be laid out in order to get an accurate estimate of its area and speed. The commonly

used method for evaluating a register-transfer level design, namely the number of functional blocks and registers and the number of control steps, is no longer adequate. That is why VLSI designers generally work from a floorplan, even when they are developing the basic system architecture.

Most ASIC vendors provide pre-layout interconnect delay estimations based on fanout and gate count, using statistical data from previous layouts. Since the lengths of wires are unknown, these pre-layout delay estimates are not accurate. One can rely on floorplanning tools to attain more accurate estimates of interconnect delay. These tools allow designers to place large macros, manipulate their size, aspect ratio, and pin positions and receive feedback on routing density and wire lengths. Based on this data, delay estimators can produce an estimated distributed RC delay. In addition to helping designers predict interconnect delays, floorplanning can provide valuable information for layout designs to reduce the number of placement and routing iterations needed.

Floorplanning tools can be used in the feedback loop of a high-level synthesis system to improve scheduling and allocation. First, a schedule and register-transfer data path are constructed. Then, floorplanning and global routing are performed which produce information about critical paths. This information is back-annotated into the circuit and high-level synthesis is repeated.

Floorplanning algorithms should model the cells' interface flexibility and any constraints on that flexibility. Three classes of cells are used in floorplanning: 1) Some cells are already laid out and are stored in a library. All the interface characteristics of these cells are known and fixed. To provide some flexibility, several versions of cells with different characteristics may be stored in a library. 2) The designs of some cells are known, but their layouts are flexible and can be influenced by the results of floorplanning. For example, standard or general cell layout methods can produce a wide range of shapes for a given design. PLAs and memory cells can be distorted through folding or layout design. 3) Cells of the third class are flexible because their designs (and perhaps even the design methods) are not known or are uncertain. In this case it is difficult for the designers or algorithms to even specify nominal interface characteristics or constraints thereon.

An important aspect of cell modeling is estimation of area and shape. This task often

requires appropriate modeling of corresponding cell generators so that cells' shape functions can be accurately and quickly estimated. The floorplanner must exploit these functions in order to trade off the sizes and shapes of the leaf cells against each other to optimize the layout.

I have developed a floorplanning procedure which is based on the bottom-up clustering, shape function computation, and top down floorplan optimization with integrated global routing and pin assignment. An important feature of my floorplanner is its ability to accept various constraints and design requirements. This is in contrast with most existing floorplanners. Since in the past, specialized conventional floorplanners have been required to handle data-path dominated assemblies, device-level analog chips, mixed macrocell and standard-cell circuits, and macrocell designs. The main difference among these layout styles is the type of constraints that must be satisfied and optimization procedures that must be applied. My floorplanner, however, provides means for specifying and techniques for satisfying a wide range of constraints (physical, topological, timing, device-level) and is, therefore, able to handle all these layout styles.

In addition to above, the floorplanner introduces the following new components:

- Accurate and dynamic routing area estimation during floorplan computation in order to avoid an increase in the chip area after global routing.
- A systematic optimization procedure during the selection of suitable floorplan patterns that integrates floorplanning, global routing and pin assignment.
- Extensions to incorporate timing issues by a novel timing-driven clustering technique, followed by top-down floorplan optimization.
- A new pin assignment technique based on linear assignment and driven by the global routing solution and floorplan topology.
- Support for some of the device-level analog floorplanning issues such as layout symmetries, matching constraints, topological constraints, cell abutting and merging.
- Tight coupling between floorplanner and module generators through modeling important classes of generators, i.e., standard cell and data path assemblies.

- Accommodation of channel-free style layout which is especially useful when floorplanning large channelless gate arrays or mixed macrocell and standard-cell assemblies.

7.2 Floorplanning Procedure

7.2.1 Overview

Prior Work

There are two major thrusts in floorplanning research. The first thrust uses floorplanning in the initial stages of design to develop constraints that can be passed to succeeding synthesis steps. The second thrust relies on the existence of powerful cell generators that can implement cells according to specifications.

Lauther's min-cut placement [Lauther 79] is a good example of the first thrust. He uses a top-down method that divides cells into two partitions (blocks). The process of dividing blocks into smaller blocks continues until the number of cells per block is small. This approach considers higher level of abstraction before it considers more detailed levels and is *goal-oriented.* Placement decisions made at higher levels of the hierarchy, however, may suffer from lack of detailed information. La Potin [La Potin 86] improved the min-cut method by considering the pad positions. Dai [Dai 87c] extended La Potin's work to general non-slicing structures and multi-way cluster trees and combined floorplanning with hierarchical global routing.

Otten's shape propagation placement technique [Otten 83] is an example of the second thrust. His algorithm initially derives a physical hierarchy in the form of a binary slicing tree. Next, the slicing tree is traversed bottom-up. At each internal node of the tree, a composite shape function is calculated from the shape functions of its children (child nodes) and directions of the cuts. These shape functions are combined and propagated recursively up the tree until the shape function for the root of the tree (root node) is obtained. An (x, y) pair on the shape function for the root node which satisfies the user specified aspect ratio is chosen and - using pointers saved during the bottom-up process - a complete floorplan is generated. Otten's approach uses a binary slicing tree which is very restrictive and

uses simplistic shape models for the leaf cells. This technique has difficulty incorporating I/O pin locations in the optimization process. Another shortcoming is that the bottom-up floorplan sizing does not consider the global connection costs. In general, this consideration is necessary because the choice of a suitable floorplan pattern for a cluster must be made on the basis of the connection cost as well.

Zimmerman [Zimmerman 88] improved Otten's technique to optimize direction of the cuts during the shape propagation phase. He estimated the wiring area required for each node of the binary slicing tree and shifted nodes' shape functions to account for the wiring areas. Herrigel [Herrigel 89] and Lengauer [Lengauer 90] proposed a top-down, follow-up phase to minimize the total interconnection length by switching cells across cuts. This method has limited effect since the layout area minimization and interconnection length minimization are separated and since the cell switching - which is performed such that the floorplan area does not change - represents small perturbation to the solution found during the bottom-up phase.

Dai et al. [Dai 89] attempted to bridge the gap between the two thrusts mentioned above by computing the best shape (i.e., a target shape) for each node of the cluster tree bottom-up and using these target shapes during the top-down floorplanning phase to evaluate the relative merits of various floorplan patterns and labelings for nodes of the tree. Single target shapes, however, do not carry enough information about the leaf cells to allow reliable decision making at higher levels of the tree hierarchy.

I extend Dai's work [Dai 89] by computing the shape functions for the cluster nodes bottom-up, incorporating constraints on the variability in cell sizes, shapes and pin positions, coupling pin assignment to global routing and finally honoring various physical, topological, timing and analog constraints.

My approach is more suitable for structured design using a top-down methodology. At the same time, because it relies on accurate bottom-up information which is calculated and stored during a pre-processing step, it does not suffer from the shortcomings of a purely top-down approach. It is also capable of generating accurate shape functions for the leaf cells from their functional or structural specifications, and produces floorplans which are good to the extent that the shape estimates are accurate.

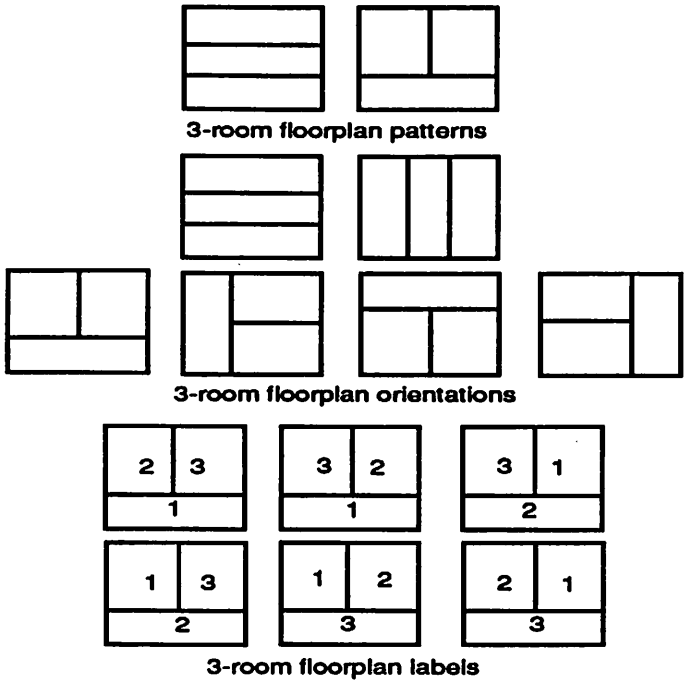


Figure 7.1: 3-room floorplan patterns, orientations of these patterns, and some of the possible labelings

Floorplanning for large channelless gate arrays can also be addressed by my approach. At the beginning of the design cycle, the gates are partitioned into a set of frames based on the timing requirements and functional hierarchy. During the physical implementation, placement improvement algorithms move cells within (but not outside of) frames with specified timing for critical paths influencing cell movement. (However, see [Murofushi 90] for a hierarchical floorplanning paradigm allowing frame overlaps.)

Terminology

A k-room *floorplan pattern* is a floorplan structure with exactly k rooms. An *orientation* of a pattern is a clockwise rotation of the pattern with respect to the boundary pin locations. A *labeling* of a pattern corresponds to the assignment of nodes of the cluster tree to individual rooms. A *topological possibility* - denoted by TP - refers to a particular choice of floorplan pattern, pattern labeling and pattern orientation. (See Figure 7.1.)

The *shape function* for a cluster node gives the lower bound on height of the node as a function of its width. The *interconnection length function* for a cluster node gives the lower bound on area of the node as a function of the estimated length of interconnections within the node. This length accounts for all wires which are required to complete connections among children of the node and from children to the external I/O pins. The combined shape function for a particular cluster node - denoted by SF_{TP} - is calculated using the shape functions for its child nodes and the TP assigned to the node. Similarly, the combined interconnection length function for a particular cluster node - denoted by LF_{TP} - is calculated using the interconnection length functions for its child nodes and the TP assigned to the node. (See Subsection 7.2.3.)

Inputs to the floorplanner are a collection of variable shape cells, shape and orientation constraints on cells, pin position constraints, locations of the chip I/O pads, a net list specifying connections among various cells and a target aspect ratio for the chip. Outputs of the floorplanner are locations, shapes and pin positions for the cells such that all constraints are satisfied and a combination of layout area, total interconnection length and aspect ratio mismatch is minimized.

7.2.2 Cluster Tree Generation

Hierarchy is the typical way to deal with the complexity of large designs. A hierarchical approach appropriately prunes the solution space and reduces the design objects to manageable sizes. In the context of floorplanning, the hierarchy usually takes the form of a tree where highly connected cells are grouped together. This hierarchical representation of cells greatly simplifies the floorplanning problem since floorplanning algorithms can recursively operate on one hierarchical cell at a time. The tree itself could have a restricted structure (e.g., binary slicing with fixed cut directions and cluster-to-room labelings) or could have a flexible structure (e.g., a multi-way tree). A more flexible structure allows for a higher degree of floorplanning optimization.

A multi-way cluster tree is, therefore, generated (Figure 7.2). This tree is obtained by minimizing connections among various cells. However, to avoid a cell shape mismatch in the clusters that makes it difficult to find a good placement for cells, shapes of the cells are

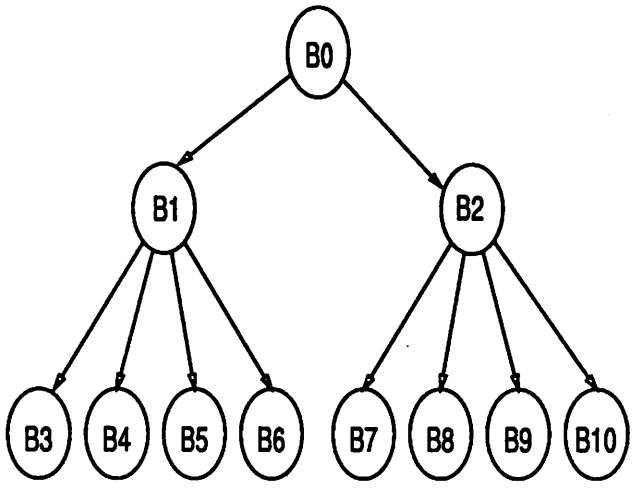


Figure 7.2: A multi-way cluster tree with 8 cells and 3 levels

also considered. The shape mismatch penalty is given more weight at the lower levels of the cluster tree, whereas at the higher levels of the cluster tree, the governing cost measure is that of the connections among various clusters. The maximum branching factor in the tree is restricted to a small value (e.g., four). This allows us to take more floorplan topologies into account that is possible with binary cut trees. At the same time, non-slicing floorplan topologies are avoided. (My approach can handle non-slicing topologies as well, however, I decided to keep away from them for reasons of efficiency.) Furthermore, if the branching factor is too large, the problem of finding a floorplan solution for a node in the tree becomes as complex as the general floorplanning problem.

To generate one level of the tree, the matching algorithm for simple graphs is used (as in [Khellaf 87]). The clustering algorithm finds a maximum weight perfect matching M_1 of G and constructs G' by contracting all edges of M_1 using the updated weights. Then, it finds a maximum weight perfect matching M_2 of G' and forms p disjoint clusters of 4 vertices each by grouping end vertices of edges of M_2. These operations are repeated recursively until one cluster node remains. The cell shape mismatch is used as a "soft" constraint, that is, only macros whose area and shape satisfy shape matching constraints become candidates for merging into the same cluster. (A soft constraint is honored as long as there is at least one feasible solution satisfying that constraint. It is ignored otherwise.)

The above cluster generation procedure is driven by connectivity and shape matching mea-

sures. It is, however, possible to derive the cluster tree directly from the architectural or logical considerations. For example, a designer may decide to put a set of macrocells in a cluster for timing or power distribution reasons. One advantage of my floorplanning procedure is that it will place and size the macros *preserving* the clusters.

7.2.3 Shape Function Computation

Each general cell has a shape function that defines its height as a function of its width. Assuming a binary tree, [Stockmeyer 83, Otten 83] showed how the combined shape function for each internal node is calculated. Briefly, the shape function for a node n_0 is found by taking the shape functions for its two children n_1 and n_2 and considering all possible combinations of an element from shape function for n_1 and one from shape function for n_2. Some of these combinations are inferior to other combinations and hence can be discarded. The way in which the dimensions from the two child nodes are combined depends on the direction of the cut line. If the cut line is horizontal, the x-dimension of n_0 is the maximum of the x -dimensions of n_1 and n_2 and the y-dimension of n_0 is the sum of the y-dimensions of n_1 and n_2. Conversely, if the cut is vertical, the x-dimension is the sum and the y-dimension is the maximum. As the shape function for n_0 is constructed, adequate routing space is added to accommodate the connections. (See Figure 7.3.)

Zimmerman [Zimmerman 88] extended the shape computation procedure to a binary node (with unspecified cut orientation). Briefly, to obtain the combined shape function for an unoriented binary node, the lower bound of the shape function corresponding to a horizontal cut and that with a vertical cut is calculated. The routing area is estimated and added to the shape function. (See Figure 7.4.)

This procedure may be extended to unoriented multi-way cluster trees as follows. Consider a cluster node with k children to which a *TP* has been assigned. Assume that this *TP* corresponds to a slicing structure[1], thus, there is a unique binary decomposition tree which represents it. The leaves of this binary tree are children of the cluster node and the internal nodes of the binary tree define directions of the cuts in the *TP*. The combined shape function

[1] For a non-slicing *TP*, the problem of calculating the combined shape function is NP-complete [Garey 79]. Either an approximate [Wong 89a] or a branch-and-bound [Wimer 88] technique can be used to do this calculation.

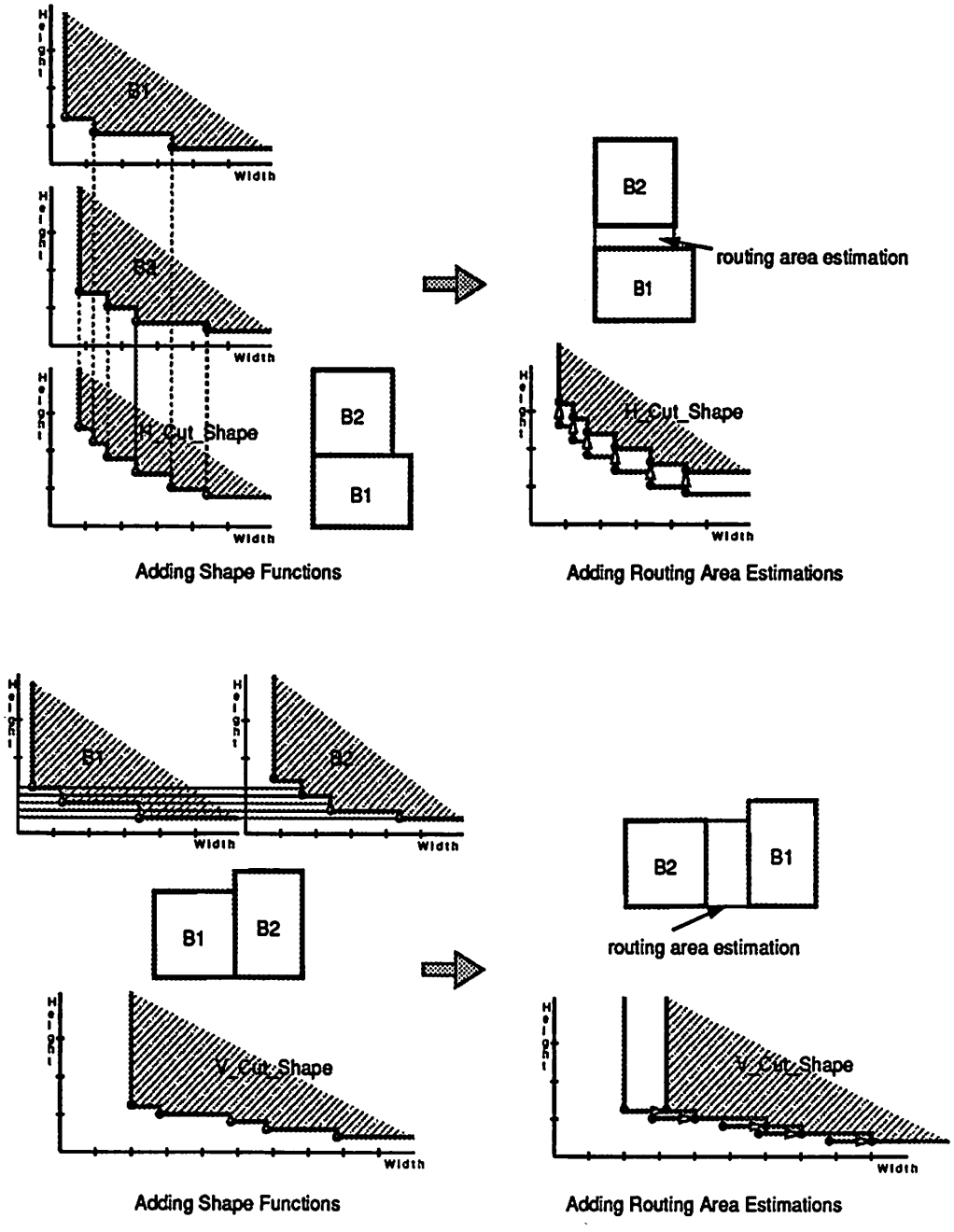


Figure 7.3: Adding shape functions for horizontal and vertical cuts

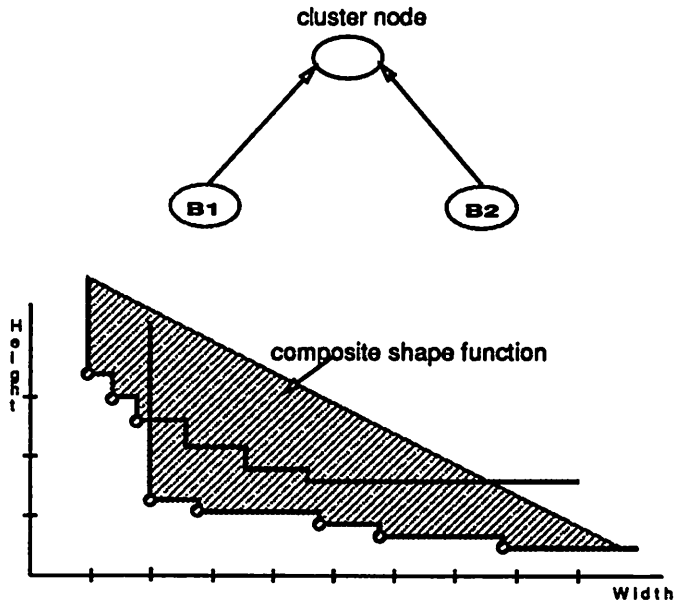


Figure 7.4: Lower bound merging of the shape functions corresponding to horizontal and vertical cuts

for the root of this binary subtree is calculated as described above.

Clustering phase, however, generates a multi-way cluster tree without assigning *TP*s to cluster nodes. The shape function for an unoriented multi-way cluster node is thus computed as follows. All *TP*s with k rooms are examined, and their corresponding shape functions are computed. For each *TP*, the routing areas around the child nodes are estimated. Each (x, y) pair on the SF_{TP} is thus shifted up and to the right to account for the wiring area required. (This is in contrast with [Zimmerman 88] which shifts the whole shape function for the node in order to account for the routing area.) Next, the lower bound of all SF_{TP}s is taken to obtain the shape function for the multi-way cluster node. (See Figure 7.5.) (x, y) pairs on the shape function are marked to represent the chosen *TP*.

This procedure is recursively applied up the tree until the composite shape function for the root node is calculated. The bottom-up shape function and a user specified aspect ratio can be used to generate a minimum area floorplan by propagating the associated geometries of the floorplan solution to the leaf nodes. The above procedure minimizes the layout

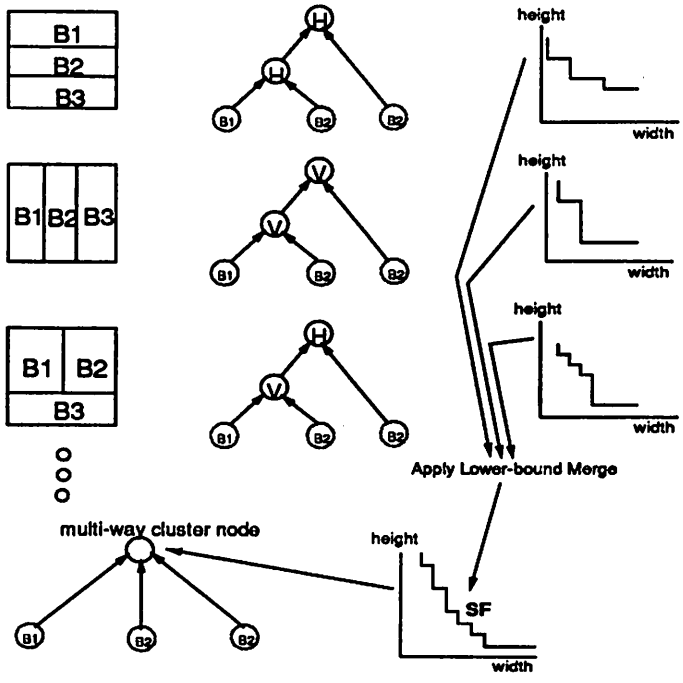


Figure 7.5: Calculation of the combined shape function for a cluster node with 3 children

area - to the extent that the bottom-up wiring area estimation is accurate - but does not optimize the total interconnection length or positions of the signal pins on flexible cells. In Subsection 7.2.4, I will describe a top-down floorplan optimization procedure which uses the bottom-up shape functions for its cost evaluation, and at the same time, searches for a floorplan solution minimizing interconnection length and satisfying timing and topological constraints. However, I shall first explain how the interconnection length functions for internal nodes of the cluster tree are computed.

The interconnection length function for each cluster node is computed as described below. For each (x, y) pair on the SF_{TP}, an $(l, a = x \times y)$ pair is calculated as follows. The list of nets *intersecting* (i.e., having at least one pin inside) the cluster node is known. For each such net, a minimum rectangle which touches all the flexible blocks and goes through the centers of the fixed blocks is constructed. (Here, flexible blocks refer to sub-clusters or variable shape leaf cells within the node, and fixed blocks refer to fixed shape leaf cells.) The idea is that since the pin positions on flexible blocks can be optimized, the minimum rectangle must only touch them at some point. However, since the cell orientations are not known during the bottom-up calculation, the center points of the fixed cells are used. Next,

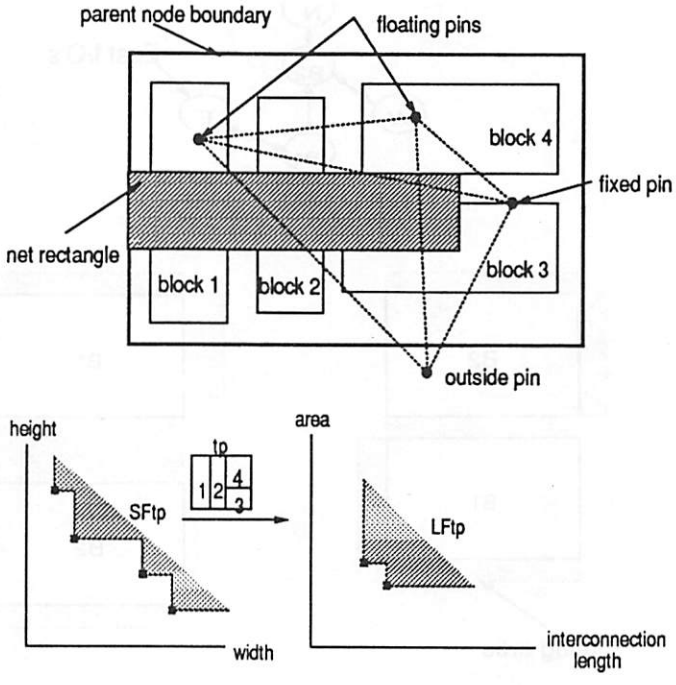


Figure 7.6: Calculation of the interconnection length function for a cluster node

if the net has pins outside the node boundary, the minimum rectangle is extended to touch the closest side on the node boundary. The half perimeter length of this new rectangle gives an estimate on the interconnection length required to complete the routing of the net in question within the node. These half perimeter lengths are summed over all intersecting nets to get l. Next, for each (x, y) pair on the SF_{TP}, an (l, a) is calculated, the inferior (l, a) pairs are dropped and the LF_{TP} is stored at the node. (See Figure 7.6.)

Again, lower bound merge operation is used to compose the successive LF_{TP}s into the composite interconnection length function for the node. This step is repeated recursively until the composite interconnection length function for the root node is calculated.

7.2.4 Floorplan Optimization

Each cluster node is floorplanned in a breadth-first manner starting from the root node. When floorplanning a node, the node aspect ratio and the external I/O pin positions are known. (These are the user specified aspect ratio and the chip I/O pads for the root node.

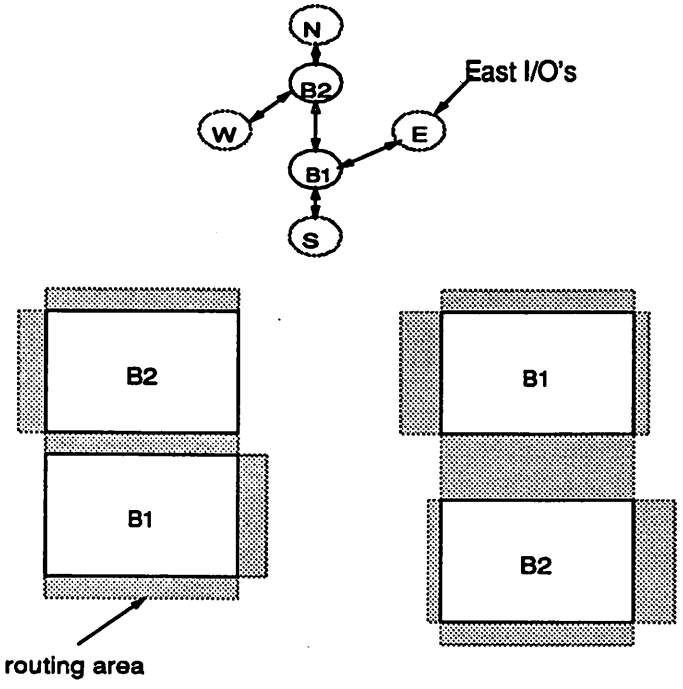


Figure 7.7: Effect of I/O connections on the routing area estimation during top-down floorplanning

For the non-root nodes, this information has been passed down the tree.) All *TP*s for the node are enumerated, and a floorplanning solution which has the minimum objective function value is selected. The objective function is a combination of area cost, interconnection cost and aspect ratio mismatch cost. Given a cluster node and a particular *TP*, the area and interconnection length costs are calculated by summing the area and interconnection length estimates for its child nodes plus the area and interconnection length required to combine the child nodes into the enumerated *TP*. (The shape and the interconnection length functions of the child nodes act as estimates of the expected layout cost from the partial layout solution where the child nodes are not yet floorplanned to the complete layout.) In Figure 7.7, the two *TP*s shown are equivalent in terms of area and wire length during the bottom-up cost calculation. However, after considering the global connections, i.e., during the top-down search, these *TP*s have different areas and wire length costs. This is exactly why a purely bottom-up floorplanning procedure is incapable of producing high quality floorplan solutions.

As a result of floorplanning a cluster node, shapes, locations and orientations of its child

nodes will become known. Positions of the boundary pins on the node, the estimated routing area around the child cells and the local net list are also known. However, I/O pins for the child nodes must be assigned positions. This I/O pin computation is necessary in order to influence the floorplanning of the child nodes by the outside connections as well as the internal connections. The task is to assign pins of the nets intersecting the cluster node to the appropriate locations on the boundaries of the child nodes such that the total interconnection length within the cluster node is minimized.

In a manner to be described in Section 7.3, the I/O pins are propagated to the boundaries of the child nodes. This procedure, therefore, sets the external I/O pins for next lower level of tree hierarchy and directly influences the choice of floorplan topology for the child nodes. At the leaf level, this I/O pin computation coincides with the pin assignment for general cells.

It is worth noting that a designer may partially or fully specify the hierarchical structure. In the latter case, he or she may not only specify the cluster tree but also assign a topological possibility to each internal node of the tree. The floorplanning problem is then to place the macros and assign shapes and pin positions to the flexible ones. The floorplanner handles such input specification. This is useful since in some cases, due to timing constraints or sensitive design, the designer may want to ensure a particular topological relationship between blocks.

7.2.5 Area Estimation

Figure 7.8 shows why it is useful to include routing area *during* rather than *after* placement. A placement which was optimized to have a rectangular shape of a given aspect ratio is probably neither rectangular nor does it conform to the desired aspect ratio after the routing area was added.

There are some approaches that with different degrees of sophistication tackle the problem. In [Sechen 85, Wipfler 85] heuristics are applied to each block in order to derive a hypothetical block shape including some routing area based on the number of pins of the block. In [Chen 84] a pseudo-routing of pairs of blocks is performed to compute spacing between two blocks in a row-based placement. Both solutions fail to account for connections outside

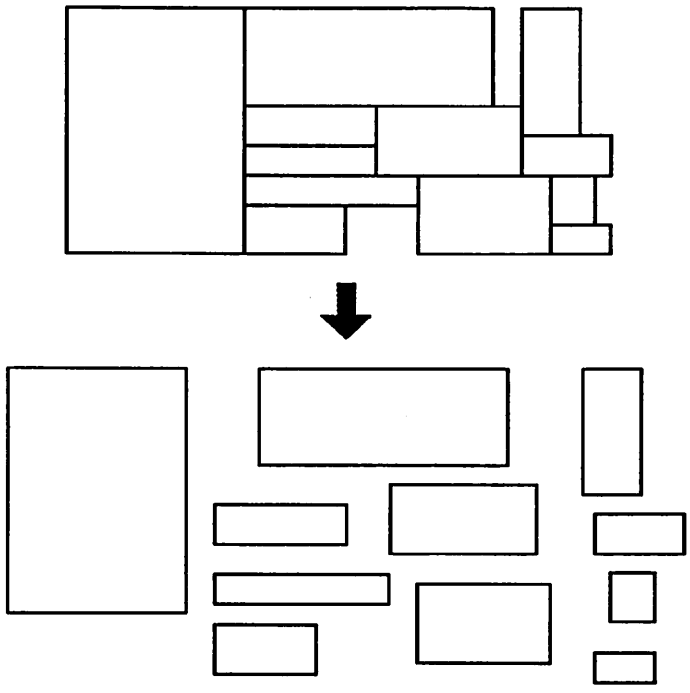


Figure 7.8: Routing area estimation after floorplanning may lead to poor results

the immediate neighborhood of the block terminals.

For gate arrays, [Burstein 83] introduced an algorithm that merges placement and routing in a hierarchical fashion. Because of the simple array structure, this grid-based approach is feasible. In [Szepieniec 86], a procedure for simultaneous placement and global routing of restricted slicing structures was proposed. A method to generate global routing at the same time as placement suited to the more general topologies was described in [Dai 87a]. In both cases no attempt was made to estimate the routing area based on the global routing information. Other systems [Lauther 79, Fowler 85, La Potin 86] perform global routing after placement is finished and then estimate the necessary routing area.

Top-down Routing Area Estimation

Besides using the hierarchical decomposition of the problem, the basic idea is to avoid a dynamic shortest path or Steiner tree determination by precomputing the paths for the finite number of floorplan patterns and storing the information in the library of patterns

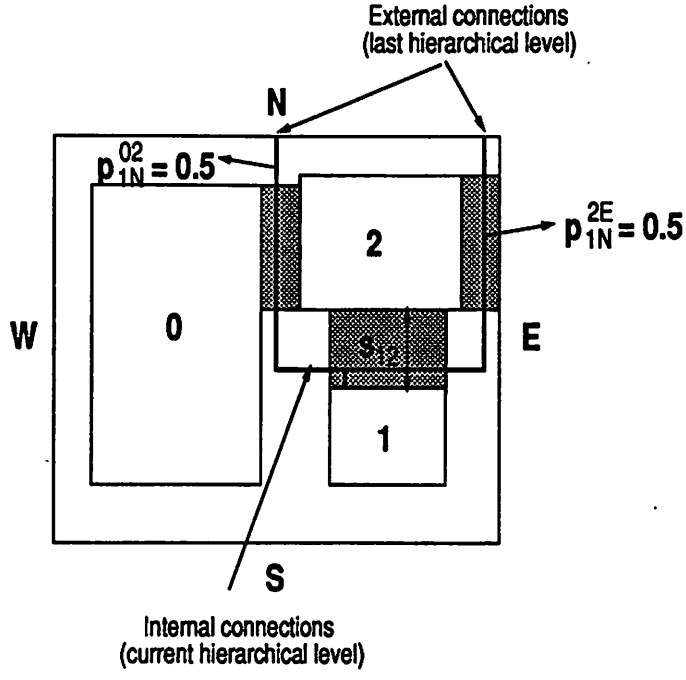


Figure 7.9: Calculation of entries in the probability matrix used by the area estimation procedure

(or *templates*).

For every template and for each connection between blocks, clusters and I/O-goals, all the channels on the shortest topological paths are marked with a probability. This probability represents the likelihood that the connection will really pass through that channel [Dai 89]. (See Figure 7.9.) p_{ij}^{kl} is the probability that a connection between blocks i and j will pass through the channel formed between blocks k and l. The amount of memory to store the library of parameters is acceptable.

The required normalized "channel"[2] width s_{kl}/w is estimated as

$$\frac{s_{kl}}{w} = t_{kl} \sum_i \sum_j p_{ij}^{kl} c_{ij}$$

where w is the design-rule dependent track to track spacing, c_{ij} is the pertinent element of the connectivity matrix and t_{kl} a heuristic factor that accounts for track sharing. (Segments of different nets share the same track.) Channel width is computed for every possible

[2] "Channels" on higher levels consist of many real channels.

topology and on all the hierarchical levels before the placement cost function is evaluated. Therefore, routing area and block area are treated in the same way. Routing area not only influences the choice of templates on the current level of the hierarchy but also sets the shape goals for the next level.

The estimation takes advantage of the information gathered about positions and connectivities of clusters (of blocks) down to that level of the tree hierarchy. Earlier in the process space for global connections between different clusters is provided. Later more of the internal connections within the clusters become visible.

The allocation of space along the shortest path makes the job of a global router easier but does not constrain it in doing whatever is recognized as optimal after the complete topological information produced by the placement is available. In addition, this approach is very flexible. For "over-the-block wirable" cells [Ueda 85], the probabilities can be easily adjusted.

Before refining the basic idea it seems appropriate to describe how the numbers p_{ij}^{kl} and t_{kl} are derived. In general the numbers to be assigned to p_{ij}^{kl} are pretty obvious as in Figure 7.9 since most of the time only two distinct shortest paths exist. For fine-tuning statistics can be compiled that characterize the behavior of the global and detailed routers used. It is likely that different routing algorithms will yield slightly different results for the parameters. It is one of the strengths of this approach that without having to modify the program, it can be applied to different technologies and physical design processes.

On the non-leaf level, it is assumed that connections leave the clusters on the side that is closest to the end point of the connection. In Figure 7.9 this means that a connection between blocks 0 and 2 would leave the right side of block 0 and enter the left side of block 2. It is the task of the next lower level to provide the space to get to the appropriate side. On the leaf level, this is no longer possible. In this case the pin position information already needed for the determination of block orientations comes in handy. For a given block orientation additional, space has to be provided along the sides of the block to bring the wires around the block to the location that is closest to the end point of the connection.

Bottom-up Routing Area Estimation

Due to the nature of the top-down placement algorithm, the area within the cluster node which holds the wiring must have been added beforehand. Instead of just assigning the sum of the areas of the children to a parent node in the clustering tree before starting the top-down traversal of the tree, a routing area estimation has to be included as well. At this stage it is not necessary to know the paths that connections take but only the approximate *area* needed for connections.

The task of predicting routing space before invoking a floorplanner or placer has gained some attention recently [El Gamal 81b, Kurdahi 86, Chen 88, Zimmerman 88]. Unfortunately the reported results applicable to the macrocell layout style are not very encouraging. Errors of 20% (of the whole layout area, much more if the known block area is excluded) seem to be current state of the art [Chen 88, Zimmerman 88]. This much error is unacceptable and will lead to poor floorplan solutions.

Fortunately, routing space can be estimated more accurately since a bit more information is available here. After clusters on one level of the hierarchy are identified, all connections between cluster elements and from one cluster to another are known. On the lowest level of the hierarchy, the number of pins along the four sides of a block is known as well. When the shape function is calculated, corresponding to each (x, y) pair on the shape function, a reasonable way of arranging the blocks is generated as a byproduct. All this information can be used to produce a routing area estimate that exactly mirrors the more sophisticated top-down routing area estimate.

Since the exact constellation that tries to minimize the number of connections between non-adjacent blocks/clusters is not known, a worst case approach is taken by building a hypothetical connectivity matrix $\tilde{C}$ with identical connection strengths $\tilde{c_i}$ and $\tilde{c_e}$ for all intra-cluster and inter-cluster connections, respectively [Dai 89]. This means that all $\tilde{c_{ij}}$ are set to $\tilde{c_i}$ for $j \in 0, ..., k-1$ and to $\tilde{c_e}$ for $j \in$ *North, West, South, East.* (See Figure 7.9.)

$$\tilde{c_i} = \frac{2}{k(k-1)} \sum_i \sum_{j \in 0,...,k-1} \tilde{c_{ij}}$$

$$\tilde{c_e} = \frac{1}{4k} \sum_i \sum_{j \in N,W,S,E} \tilde{c_{ij}}$$

With this connectivity matrix and the shape topology, the top-down wiring space allocation function is called. The exact distribution of the wiring space generated by that function is of no interest here. The only useful extraction is the space needed for routing. The routing area (contrary to its exact allocation) is very similar for various good topologies (i.e., topologies with little dead space).

On the leaf level, block areas are inflated in both dimensions based on the number of pins of the block in each direction. In the final placement result, connections between non-adjacent blocks tend to be weaker than connections between adjacent blocks. Therefore, the initial routing area estimate is reduced by some heuristic factor. If with the default value of that factor the routing area is consistently over- or under-estimated, it can be adjusted by the user.

7.2.6 Complexity Analysis

Under the assumptions that the clustering tree is a tree with n leaf nodes, that each leaf node has a shape function with at most m shapes, and that every internal node has k children, the complexity of the floorplan optimization algorithm is given by:

$$O(d\ n\ m\ f(k))$$

where $f(k) = k! \times t(k)$ is the number of enumerated *TP*s ($t(k)$ is the number of non-isomorphic oriented floorplan patterns) and is given in Table 7.2.6. The derivation of

k	1	2	3	4
$f(k) = k! \times t(k)$	1	4	36	528

Table 7.1: Number of topological possibilities $f(k)$ for non-leaf clusters

complexity is as follows. Assume that root of the cluster tree is at level 0 and leaf nodes are at level d. A k-tree of depth d contains $n = k^d$ leaf nodes and $(n-1)/(k-1)$ internal nodes. During floorplanning, the combined shape function for each internal node of the tree is computed. In particular, for a node at level $d-1$, $f(k)$ *TP*s are enumerated where each *TP* requires $k-1$ shape function add and $k-2$ shape function merge operations. Each operation takes time proportional to the number of shapes in the shape functions. At level

d, each shape function has m points, therefore, $O((k-1)\ m\ f(k))$ is required to compute the combined shape function for a node at level $d-1$. There are k^{d-1} nodes on level $d-1$, therefore, the processing time for nodes on this level is $O(k^{d-1}\ (k-1)\ m\ f(k) \sim n\ m\ f(k))$. The number of shapes on a shape function at level $d-1$ is at most $k\ m$. For nodes at level $d-2$, the processing time is $O(k^{d-2}\ (k-1)\ k\ m\ f(k) \sim n\ m\ f(k))$. The shape functions for a node at level $d-2$ will have at most $k^2\ m$ points. By induction, it can be shown that the processing time for every level of the cluster tree is $O(n\ m\ f(k))$ and since the tree has d levels (excluding the leaf level), the desired result is derived. Note that if bucket sorting is used, that is if an upper bound, say $M \geq m$, is set on the number of shapes in the shape function for each internal node, then the algorithm runs in $O(n\ M\ f(k))$. The run time can be significantly reduced if symmetries of k-room *TP*s are exploited and / or sub-templates are shared.

7.3 Pin Assignment with Global Routing

7.3.1 Overview

Previous works on pin assignment assume that shapes and positions of cells are given as input data. These algorithms can be classified into three categories:

1 Those which assign pins on a cell by cell basis [Koren 72, Brady 84];

2 Those which assign pins on a net by net basis [Yao 88, Yao 90];

3 Those which sequentially process edges of a supergraph containing the global route solutions for all nets, finding a coarse pin assignment and global routing solution followed by a local pin assignment optimization for that global routing [Cong 89].

Among these approaches, only [Cong 89] correlates pin assignment with global routing. However, the quality of the global routing with coarse pin assignment depends on the ordering in which the 'non-essential' edges are eliminated and there is no way to determine

a 'good' ordering for edge deletion. It is not clear to us how any of these approaches could be extended to include the floorplanning task.

The technique proposed here solves the pin assignment and global routing problems simultaneously. This technique avoids difficulties associated with the cell or net ordering during pin assignment. Floorplanning determines positions and shapes of hierarchical cells, sets the channel topology and assigns capacities to the routing regions. Pin assignment and global routing operate on the hierarchical floorplanning solution and are weaved in order to produce assignments for floating pins which minimize the layout area as well as the total interconnection length. The initial pin assignment sets the stage for the global routing step by assigning positions to the floating pins. Global routing, then, determines connection patterns and defines channel densities. This information is subsequently used to adjust the pin positions. Global spacing is also performed in order to guarantee routing success.

7.3.2 The Procedure

Suppose that the root of the cluster tree has been floorplanned. In the process, children of the root have been assigned shapes, positions and pin locations. Next, these child nodes are floorplanned in the order of decreasing area. However, prior to floorplanning a child node, the global net list is updated to include cells and connections inside the node. Updating the cell list means that the node is deleted from the list and its children are inserted into the list. Updating the pin lists consists of deleting pins on the node boundary and adding pins on the cells inside the node. Referring to Figure 7.10, D is about to be floorplanned. The current net list consists of cells A, B, C, D and nets $n_1 = (p_4, p_5, i_1)$ and $n_3 = (r_1, r_2)$. The global net list is updated to include cells A, B, C, $D1$, $D2$ and $D3$ and new nets $n_1 = (p_1, p_2, p_3, p_4, p_5)$, $n_2 = (q_1, q_2)$ and $n_3 = (r_1, r_2)$. This updating is beneficial since it provides a global view of the layout plane and the connections during floorplanning and pin assignment of D.

After a node (e.g., D) has been floorplanned, shapes and positions of its child nodes ($D1$, $D2$ and $D3$), shapes, positions and pin locations for other nodes (A, B and C), the estimated routing area around the cells (channel capacities) and the global net list are known (Figure 7.11). The goal is, then, to assign locations to the I/O pins on the child cells such

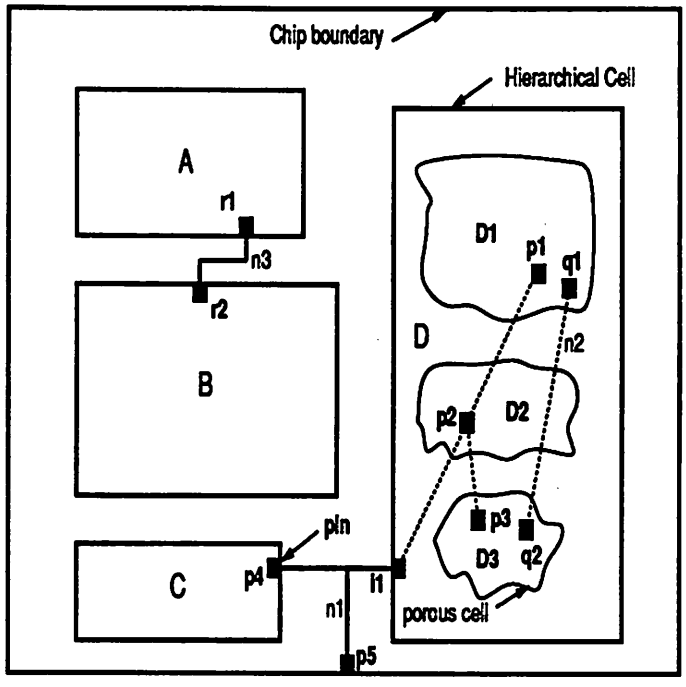


Figure 7.10: Partial floorplan solution prior to floorplanning D.

that the channel capacity constraints are satisfied while the total interconnection length and the critical net length violations are minimized .

In order to avoid necessity for the sequential processing of cells or nets, the pin assignment problem is transformed into a linear sum assignment problem as follows. A cost matrix whose rows correspond to the floating pins on the child cells and its columns correspond to the *pin slots* (feasible pin locations) on the child cells is constructed. By solving the linear assignment problem, locations for the floating pins are determined. For that assignment, global routing is performed and channel densities are calculated. If some channels are over-subscribed, the pin assignment procedure is repeated. The initial and final assignments differ only in the way that the linear assignment cost matrix is set-up and filled in.

Routing area may be very irregular. Therefore, in order to store the routing information, the general approach of [Ousterhout 84, Dai 87a] is used. The entire area of a layout is covered with rectangles referred to as *tiles*. There are two kinds of tiles: *solid* tiles which represent cells and *space* tiles which represent empty space for routing between the cells. Given a placement of rectangular shaped general cells, two tile planes are defined: the

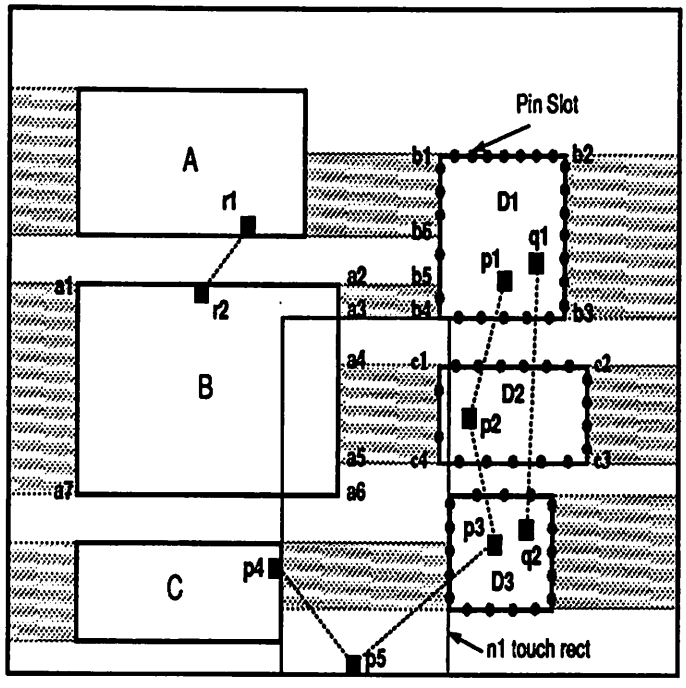


Figure 7.11: Partial floorplan solution after shape and position calculation.

horizontal tile plane where all space tiles are maximal horizontal strips and the vertical tile plane where all space tiles are maximal vertical strips. In the tile plane, each space tile has four edges: two of them are called *spans* of the tile (which are completely covered by the solid tiles); the other two form *sides* of the tile. A space tile is a *bottleneck* tile if its sides are covered by the sides of adjacent space tiles. These are areas where wire congestion is most likely to occur. A *junction region* is the maximal empty space which is completely surrounded by the solid tiles, bottleneck tiles or the plane boundaries.

Each side of a solid cell is divided into a set of *segments*. The *bottleneck* segments are those maximal intervals of sides of cells which are fully covered by the adjacent bottlenecks. The *junction* segments are the remaining maximal segments. Tile planes are shown in Figure 7.11. (a_2, a_3), (a_4, a_5) and (a_7, a_1) are the vertical bottleneck segments of the cell A. (a_3, a_4) and (a_5, a_6) are vertical junction segments of cell A. Similarly, (b_2, b_3), (b_4, b_5) and (b_6, b_1) are the vertical bottleneck segments of B and (b_5, b_6) is a vertical junction segment of B.

For each segment of each cell, the number of feasible pin slots is calculated. First, the

procedure for calculating the number of pin slots on the bottleneck segments is described. Suppose that at most t parallel wires can pass through a bottleneck. Presume that $\alpha \times t$ pins can be placed on each segment covered by that bottleneck. $1/\alpha$ is the track utilization factor which is about 0.65 using a standard channel router. The pin slots are uniformly distributed along the bottleneck segment. (The length of a bottleneck segment and the number of pin slots in it determines the minimum pin-to-pin spacing which must be at least as large as that dictated by the design rules.) Next, the number of pin slots on the adjacent bottleneck segments are summed over to give the number of pin slots on each cell. If a cell has less slots than it has floating pins, new pin slots are added on the junction segments. For each junction segment, the more pessimistic spacing of the adjacent bottleneck segments is used. For example, referring to Figure 7.11, spacing of pin slots in segment (b_4, b_5) is bigger than the spacing in segment (b_6, b_1), therefore, for junction segment (b_5, b_6), the same spacing as that in segment (b_4, b_5) is picked. If after adding pin slots to the junction segments, there are not enough slots on some cells, the node may be decompacted so that the number of feasible slots on each child cell may be increased to be equal to or bigger than the number of floating pins there.

For the initial pin assignment, the cost matrix is denoted by $[C]$ and its entries are determined as follows. For each net having floating pins on the child cells, a minimum rectangle which touches the child cells and the external I/O pins connected by the net is constructed. (Figure 7.11 shows the touch rectangle for net n_1.) All the slots that fall within this *touch rectangle* and lie on the child cells connected by the net are assigned a zero cost. Other slots have positive costs proportional to their Manhattan distances from the touch rectangle. Pin slots on the cells which are not connected by the net are assigned infinite costs. Next, a linear assignment algorithm [Burkhard 80] is run on the matrix $[C]$. Since rows in the cost matrix $[C]$ correspond to floating pins and columns correspond to the pin slots, the linear assignment determines pin assignment with the minimum cost.

During the initial pin assignment, the bottleneck congestions are only implicitly considered (by controlling the number of available slots per segment of each child node). However, chip area and total wire length can be accurately estimated only after global routing. It is, therefore, necessary to combine global routing with pin assignment as is described below. After initial pin assignment, global routing on the partial floorplan produces the shortest

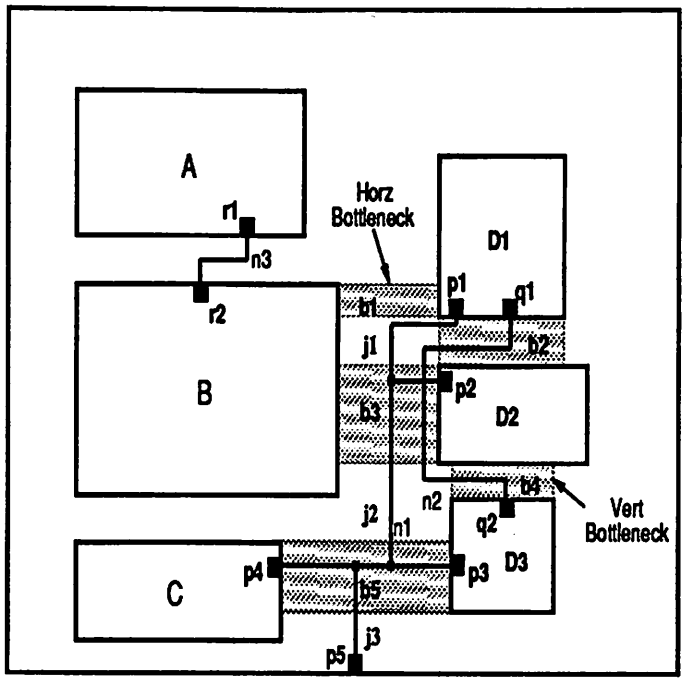


Figure 7.12: Partial floorplan solution after initial pin assignment.

connection paths for all nets. This routing scheme may result in over-congested channels. In that case, a final pin assignment which repositions the floating pins on the child cells in order to reduce congestions in the over-subscribed channels is performed. First, the number of pin slots on each segment of each child cell is re-calculated based on the bottleneck congestions after global routing. In particular, the number of pin slots in the over-subscribed bottlenecks (density > capacity) is decreased, and this number in the under-subscribed bottlenecks (density < capacity) is increased. Next, the new cost matrix $[D]$ is calculated. Its structure is similar to that of the matrix $[C]$, that is, $[D]$ has the same number of rows as $[C]$ but may have different number of columns.

For each net, the connection tree produced by the global router is examined. For this tree, the list of junction regions that the net goes through are identified. All the slots that fall within these junction regions and are on cells connected by the net have cost zero. All other slots on the connected cells have a cost proportional to their minimum Manhattan distances from the nearest junction region. Slots on cells which are not connected by the net have infinite cost.

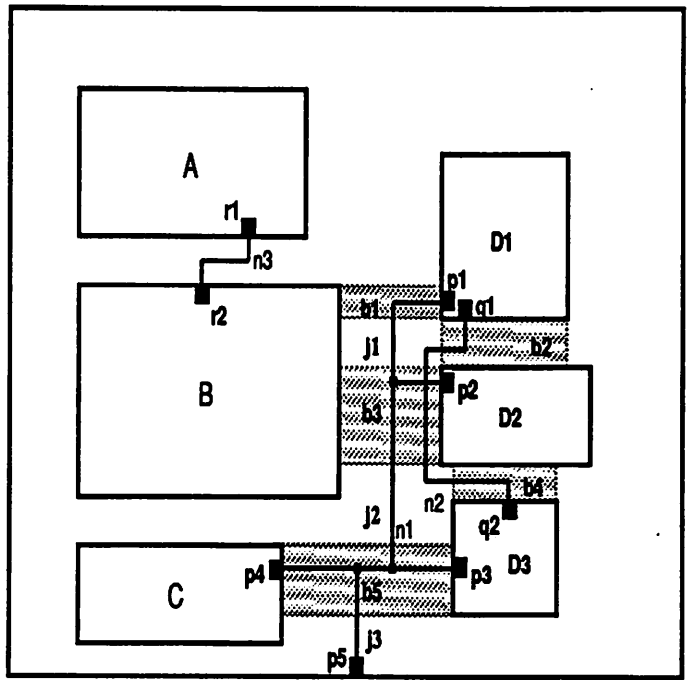


Figure 7.13: Partial floorplan solution after final pin assignment.

To motivate the above slot cost calculation, consider Figure 7.12. This figure shows the partial floorplan solution after the initial pin assignment and global routing on Figure 7.11. The global routing for n_1 goes through junction regions j_1, j_2 and j_3 and for n_2 goes through j_1 and j_2. Without loss of generality, assume that bottleneck region b_1 is under-subscribed and b_2 is over-subscribed. The goal of the second pin assignment is to alleviate routing congestion in b_2 by moving pins out of that region. To achieve this goal, the number of pin slots in b_2 is reduced, and the number of pin slots in b_1 is increased. Therefore, there will be more competition (among pins of competing nets n_1 and n_2) for available pin slots in b_2 and less competition for those in b_1. Since there are not enough pin slots in b_2 to accommodate all the pins, pins of some nets have to be shifted out. There are pin slots in b_1 and b_2 which have the same Manhattan distances from the junction region j_1. Therefore, either p_1 or q_1 can be moved into b_1 without increasing the sum cost. No pins in b_1 has to move out since the number of slots in b_1 have been increased. Thus, the linear assignment solver will move either p_1 or q_1 out of b_2 into b_1 (Figure 7.13.) In general, this cost calculation procedure tends to reduce the channel congestions with a minimal increase in the total interconnection length.

It is worthwhile noting that the above procedure based on linear sum assignment does not find the optimal pin locations within each routing channel, and therefore, must be followed by a channel pin arrangement procedure as in Section 7.5. For example, consider net n_3 in Figure 7.13. This net does not pass through any junction region. It is advantageous to minimize the half perimeter length of the box enclosing its pins. However, this task cannot be accomplished using linear assignment since the cost of assigning a pin to a slot is dependent on the position of the other pin.

This method also handles nets whose pins were preassigned at the expense of more work during the floorplanning phase and slightly more complex processing during the slot generation and positioning phase. In particular, during the floorplanning step, orientations of the cells must be optimized based on the locations of their fixed pins, and when calculating the number and the distribution of pin slots within bottleneck boundaries, the presence of fixed pins is taken into account. (A list of free boundary regions for each child node is maintained.) One may wish to have a special pin assignment for power and ground nets to satisfy planar routing topology for these nets. In one such scheme, all *Vdd* pins are placed on pin slots located on the top and left cell boundaries and all *Gnd* pins are placed on bottom and right boundaries by giving infinite cost to undesirable pin slots for each *Vdd* or *Gnd* pin. If there are some critical nets, pin slots which are located outside the zero-cost regions for the nets are assigned very high costs, hence, ensuring minimum interconnection lengths for the critical nets. Of course, this may lead to increased wire length for non-critical nets and increased total wire length.

Feedthroughs can be inserted on the non-leaf nodes. After constructing the minimum touch rectangle for a given net, if the rectangle is completely 'blocked' by a child node, two feedthrough pins are inserted on the child node. Next, the net is decomposed into two spanning subtrees as follows. A minimum spanning tree connecting all pins of the net (which must include the feedthrough pin(s) and the edge that goes through the child node) is constructed. The feedthrough edge is subsequently removed and two connected subtrees are left. The pins in each subtree define a subnet which is then passed to the pin assignment and the global routing steps.

7.4 Shape Optimization

7.4.1 Overview

The initial floorplanning is followed by the global routing process which defines densities of the routing areas. After global routing, the global spacing procedure assures that all bottleneck tiles have capacities equal to or exceeding their corresponding densities.

A global shape optimization phase follows next. This is useful because after global routing and spacing, the channel densities are likely to change, and hence, it is possible to decrease the chip area by reshaping some of the flexible blocks.

One-dimensional and two-dimensional shape optimization algorithms have been implemented. The one-dimensional algorithm iteratively computes new dimensions for flexible blocks in order to reduce the chip dimension in the user-specified *resize direction* (horizontal or vertical). This algorithm does not change chip dimension in the direction orthogonal to the resize direction. The two-dimensional algorithm iteratively reduces the layout area by picking up a block with the largest resize possibility and resizing it. Relevant terminology can be found in Subsection 7.3.2.

Block adjacency graphs [Dai 87b] can be used to calculate the extents of the chip. The longest or *critical* paths through the horizontal and vertical block adjacency graphs determine width and height of the layout. Vertices of the horizontal block adjacency graph represent blocks and arcs represent horizontal bottleneck tiles. Weights on vertices are horizontal dimensions of the corresponding blocks and weights on arcs are densities of associated bottleneck tiles. The vertical block adjacency graph is defined similarly. Bottleneck densities (rather than bottleneck capacities) are used since longest paths computed using bottleneck densities give more accurate estimations of the post-layout chip dimensions. In addition, because the global routing information is incrementally updated and bottleneck densities recomputed from one iteration to the next, longest paths through the layout surface remain accurate and representative of the final chip dimensions.

7.4.2 The Procedure

The estimated width and height of the chip at the jth iteration are denoted by W and H respectively. If block i with width w^i, height h^i and area a^i does not belong to the longest path in the current optimization direction, it will have some freedom to move or deform in that direction without enlarging the chip area. Considering the X-direction, the legal X-slack of this block $x^i_{legalSlack}$ is determined as follows: Assume that the length of the longest path from the left boundary of the chip to the left boundary of the block is l^i and the length of the longest path from the right boundary of the block to the right boundary of the chip is r^i. Then, the X-span of the block (horizontal range where the block can be placed without overlapping other blocks or causing overflows in the neighboring bottleneck tiles) is given by

$$x^i_{min} = l^i$$

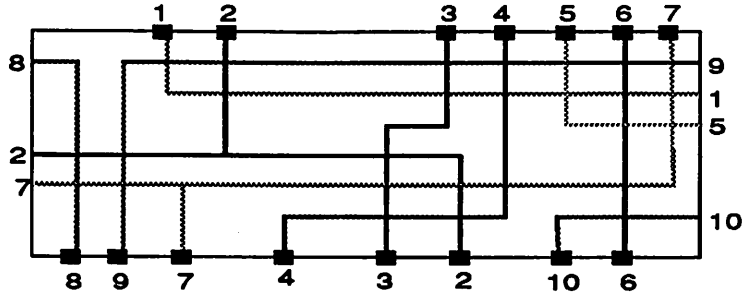
$$x^i_{max} = W - r^i.$$

The lower left corner of the block may be positioned between x^i_{min} and $x^i_{max} - w^i$ or its width may be increased by an amount $x^i_{slack} = x^i_{max} - x^i_{min} - w^i$. The two operations can be combined without enlarging the chip dimension in the X-direction. Now, suppose that the block height has a lower bound h^i_{lower} and its width has an upper bound of w^i_{upper}. Then

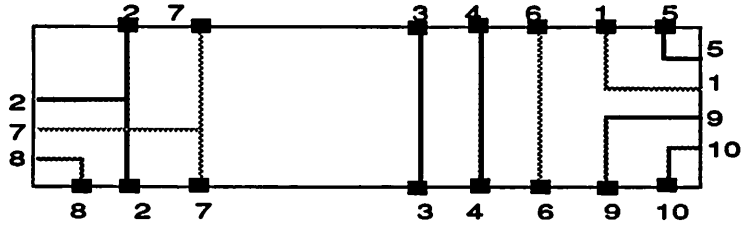
$$x^i_{legalSlack} = Min(x^i_{slack}, w^i_{upper} - w^i, \frac{a^i}{h^i_{lower}} - w^i).$$

Similarly, the legal Y-slack $y^i_{legalSlack}$ of block B^i is defined as a function of the upper bound h^i_{upper} on its height and the lower bound w^i_{lower} on its width.

Initially, the longest paths through the layout in X- and Y-directions are computed. Next, a block B_i which lies on the longest path in one direction (e.g. Y-direction) and has the largest legal slack, $x^i_{legalSlack}$, in the other direction (e.g. X-direction) is selected. This block may be laterally shifted (in X-direction) and/or resized by $\Delta w^i = \gamma x^i_{legalSlack}$ where $0 < \gamma \leq 1$ is a user specified parameter bounding the maximum change in block dimensions per iteration. The new block dimensions are $\hat{w}^i = w^i + \Delta w^i$ and $\hat{h}^i = a^i/\hat{w}^i$ if B_i is a general cell. If B_i is a standard cell or data path block, $\hat{w}^i$ and $\hat{h}^i$ will be rounded to the closest (x, y) pair in the block shape function. The new position of block is adjusted in order to distribute the 'free' area among the surrounding routing channels according to their wiring



Before Detailed Pin Arrangement (6 tracks)



After Detailed Pin Arrangement (4 tracks)

Figure 7.14: Effect of channel pin arrangement procedure.

demands. The global spacer may be called to assure that no surrounding bottleneck tiles will overflow as the result of block resizing. The global routing around the block is locally updated. After block B_i is resized, the chip height is usually decreased but the chip width remains relatively unchanged. In general, the stopping criterion is that longest paths only contain fixed size blocks (or maximally warped flexible blocks) or contain flexible blocks which lie on the longest paths in both X- and Y- directions.

7.5 Channel Pin Arrangement

The unconstrained channel pin arrangement problem is to determine the pin positions on the bottom and top edges of a routing channel such that the resulting channel density is minimum. In [Cong 91] a near-optimal solution for the unconstrained problem is presented. The solution technique, however, places pins of the same net which are on the same side of the channel next to each other. Although the channel density is then nearly minimized, the produced pin arrangement solution is far from desirable. In fact, in many cases, it is

desirable to have pins of the same net which are on the same side of a macro-cell be placed far from one another, because the signal must be sent to different regions on that macro-cell.

The generalized channel pin arrangement problem imposes position and partial order constraints on the pins. In [Cai 90], it is shown that the generalized channel pin arrangement problem is NP-complete and a polynomial time algorithm is presented for the case of a linear ordering with no position constraints.

Currently, I use a heuristic procedure for assigning pins on the sides of the channel which is similar to that in [Cong 91]. This procedure is effective since in most of the examples, nets have at most 2 pins (on opposite sides) and 2 exits in any routing channel. (See Figure 7.14.)

7.6 Analog Placement Issues

Today's VLSI chips often contain whole systems including the interfaces with analog inputs or outputs. The design time and cost associated with the dedicated analog interface modules often constitute a bottleneck in semicustom design of mixed analog / digital systems. That is one reason for the recent interest in automatic layout tools for analog devices.

Layout of an analog circuit strongly influences its performance. Performance constraints imposed on an analog circuit are often converted into physical constraints such as symmetry, matching, and equal distances between pairs of modules. An analog module has a number of different implementations for its shape; its pin positions are, however, fixed for each implementation. These physical constraints can be easily incorporated into the floorplanning procedure by suitable modifications to cluster tree generation procedure followed by appropriate pruning of the search space during the top-down floorplan optimization phase.

Assume that the analog designer provides the physical constraints. Automatic generation of physical constraints from performance objectives is not addressed here. Therefore, the following classes of constraints are considered during the placement process:

- **Symmetry** — Only half of the modules are considered during the placement. The other half will be the mirror image of the placed modules with respect to a symmetry line.

- **Matching** — Modules with matching constraints are forced to assume similar orientations.
- **Merging/Abutting** — During clustering, a matched pair of modules are merged into a single module (cluster).
- **Net Length Bounds** — Upper bound length constraints for nets is honored.

From the above list, items 1 and 2 are straight-forward and will not be discussed further. Item 4 will be addressed as part of the net-based performance-oriented floorplanning procedure. Item 3, however, needs further discussion. If modules with matching constraints belong to the same cluster, the problem is easy since during the top-down enumeration phase, all different orientations for this group of modules will be considered and the best is chosen. However, if the modules belong to different clusters, then their orientations can not simultaneously be optimized and this will introduce an ordering dependency in the sense that the orientation of the first module fixes the orientation of all other modules in the matching group. Hence, firstly modules in a matching group are to be assigned to as few clusters as possible and secondly, the cluster with largest number of modules in any matching group must be processed before others.

Other analog issues (such as crosstalk avoidance, over-the-device wiring, well merging, bulk contacts, symmetric routing, transmission line effects and so forth) must be considered by specialized analog routing systems. (See [Garrod 88, Choudhary 90].)

7.7 Experimental Results

Due to lack of published results on floorplanning and pin assignment, comparative results for BEAR-FP floorplanner could not be presented. Therefore, I ran BEAR-FP on *Xerox*, *Ami33* and *Ami49* general cell benchmarks and recorded layout area and interconnection length after routing (for chip aspect ratios 1.0, 2.0, and 4.0). Next, I ran BEAR-FP on the variable-shape version of these benchmarks. For *Xerox-F*, I used the shape list specified in [MCNC 90]; for *Ami33-F* and *Ami49-F*, I generated 7 shapes per macrocell (spanning aspect ratios 1.0 to 3.0 with constant area). Pins on the fixed-shape macrocells were fixed,

and pins on the variable-shape macrocells were completely floating (no side, position or ordering constraints). Cell rotation was permitted. Table 7.2 summarizes characteristics of these benchmark circuits.

example	numCells	numNets	numIOs	shapeType	pinType
Xerox	10	203	2	F	F
Ami33	33	121	38	F	F
Ami49	49	408	22	F	F
Xerox-F	10	203	2	V	L
Ami33-F	33	121	38	V	L
Ami49-F	49	408	22	V	L

Table 7.2: Description of benchmark circuits (F = Fixed, V = Variable, L = fLoating)

Table 7.3 presents the results. The chip area and total wire length for the variable-shape benchmarks are about 10% and 15% less than their fixed-shape counterparts respectively. The run time (on DEC3100) for the *Xerox* and *Xerox-F* circuits are 17 and 43 seconds. Most of the time is spent in the hierarchical pin assignment and global routing steps.

	aspect ratio = 1		aspect ratio = 2	
example	chip area	wire length	chip area	wire length
Xerox	27.2	626.1	25.6	613.8
Ami33	2.65	131.5	2.69	132.3
Ami49	50.6	983.3	52.3	994.1
Xerox-F	26.1	540.4	25.1	535.3
Ami33-F	2.34	109.7	2.45	110.6
Ami49-F	45.2	713.4	48.0	721.6

Table 7.3: Chip area (mm^2) and total wire length (mm) for the benchmark circuits

Figures 7.15 and 7.16 show the pre- and post-routing results for *Ami33* benchmark. Figures 7.17 and 7.18 show the results for *Ami33-F* benchmark. There is good match between the top-down routing area estimations and the final routing areas.

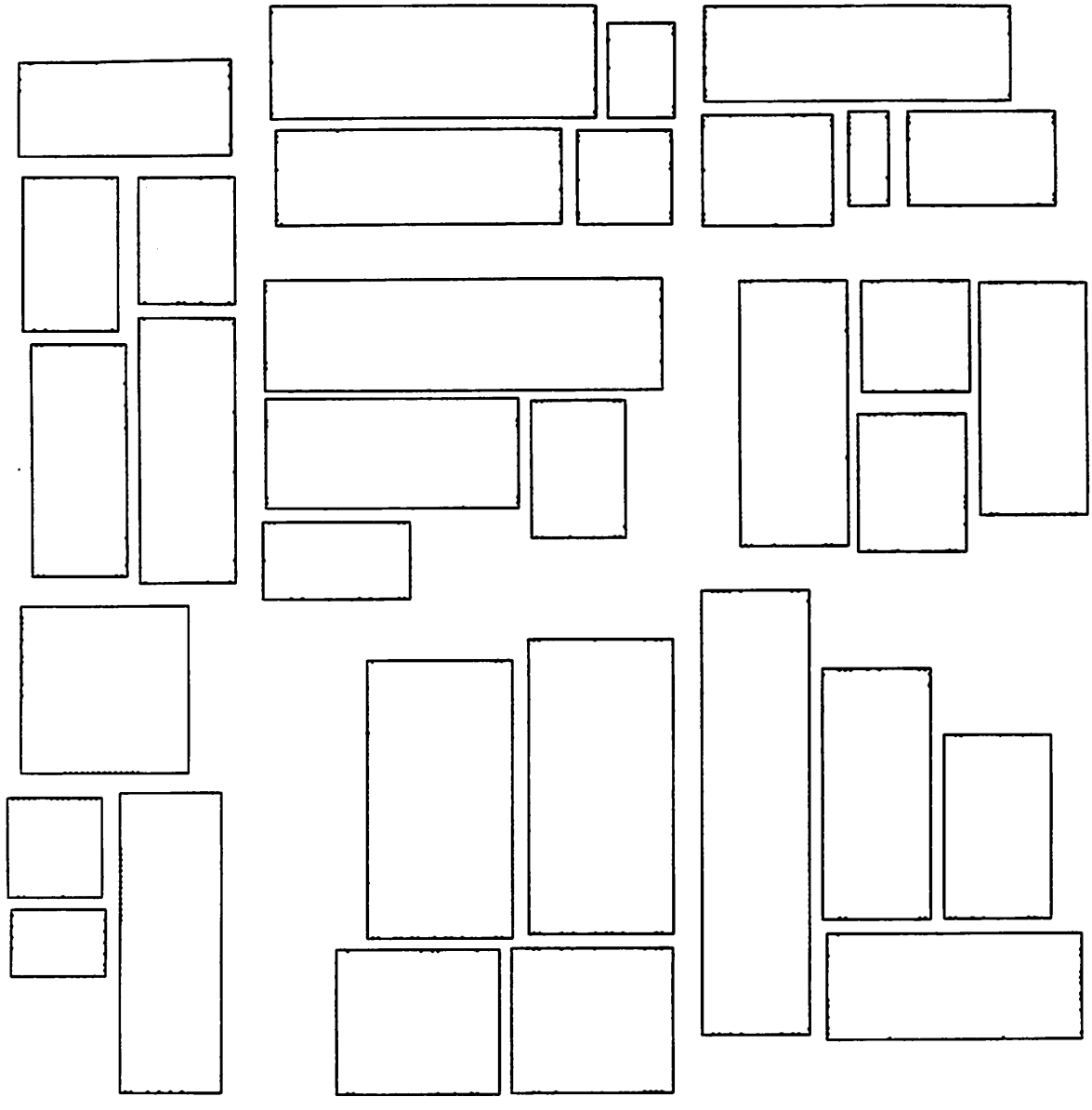


Figure 7.15: Placement result for *Ami33* benchmark

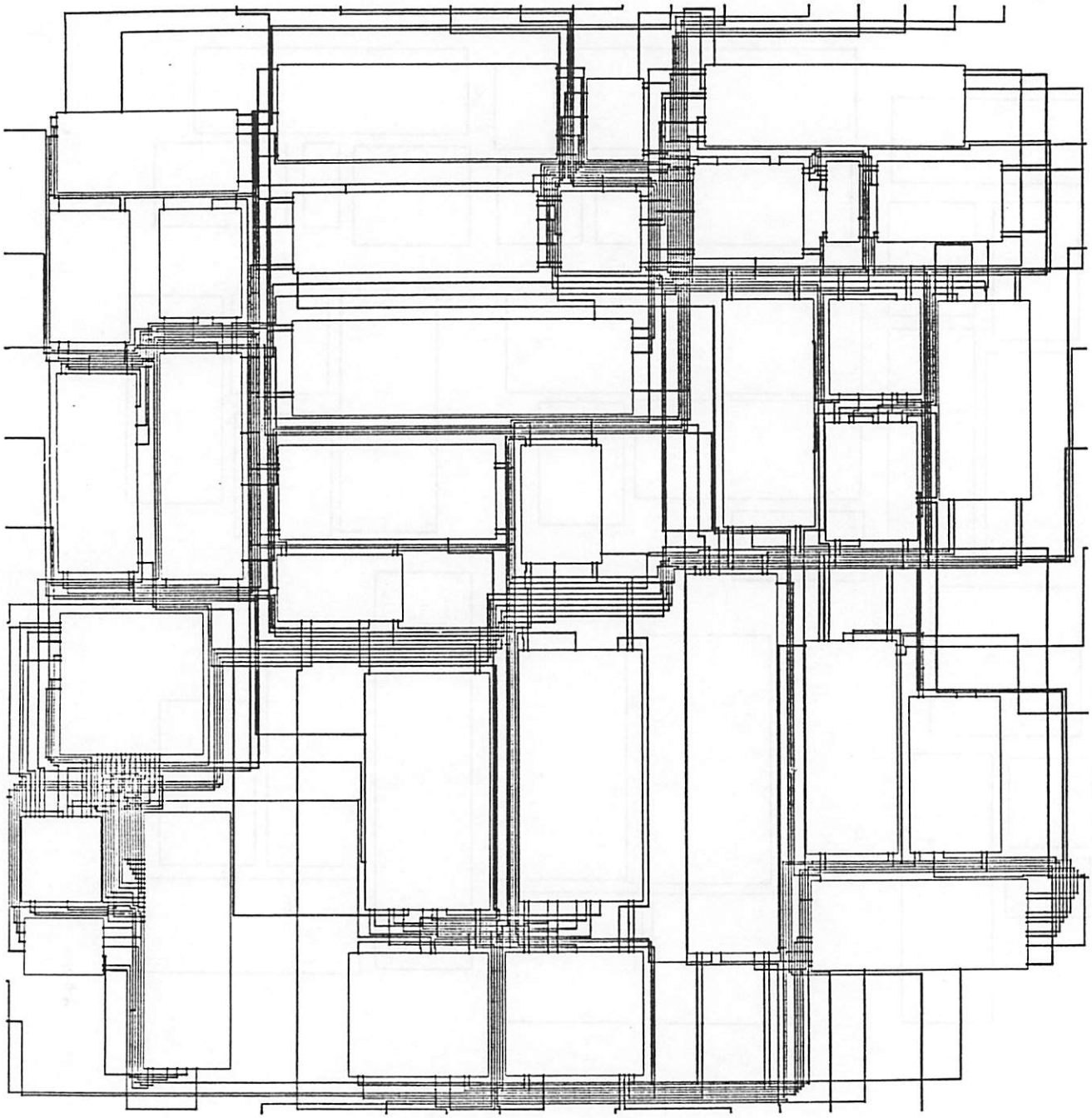


Figure 7.16: Routing result for *Ami33* benchmark

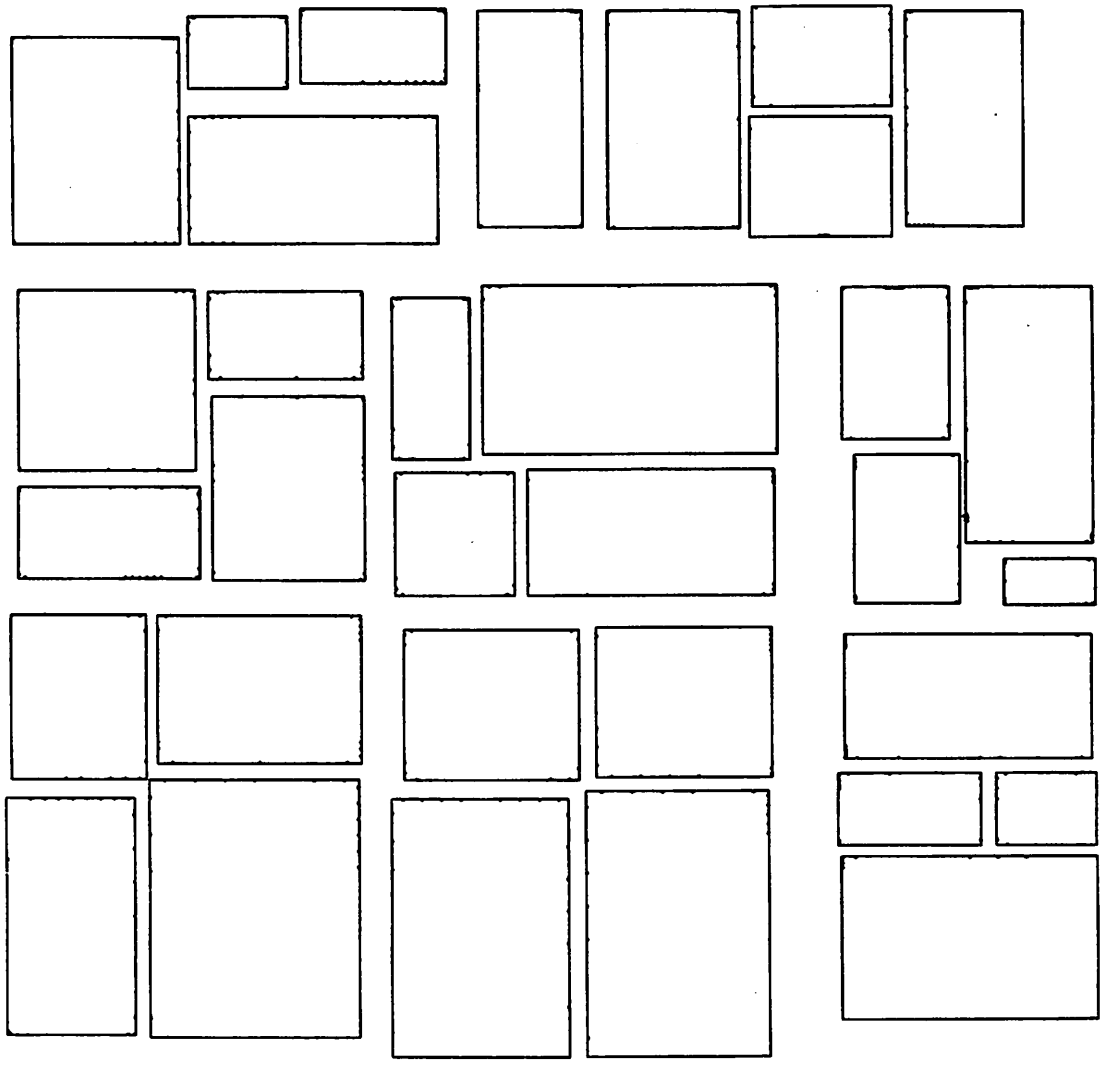


Figure 7.17: Placement result for *Ami33-F* benchmark

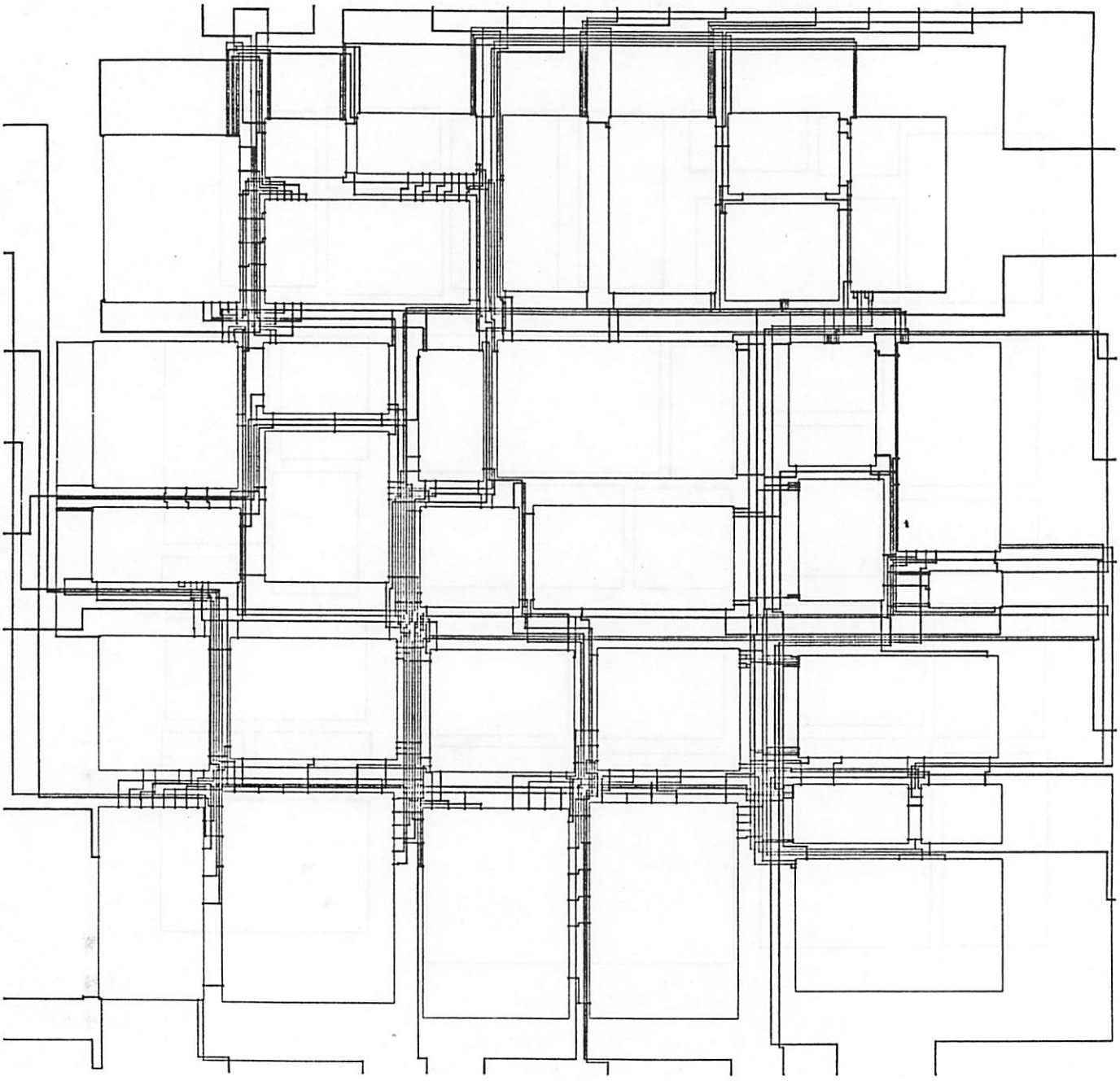


Figure 7.18: Placement result for *Ami33-F* benchmark

Chapter 8

Performance Oriented Floorplanning

8.1 Introduction

As IC fabrication technology improves and performance requirements on designs increase, it is becoming essential to explicitly optimize the chip performance. Furthermore, since the contribution of interconnection length to the overall chip delay is increasing, automatic control of interconnection length is indispensable. To meet the needs of an expanding electronic industry, high-performance chips must be designed in a short period. Accordingly, a straight forward design flow which incorporates timing analysis and verification into the the physical design process is desirable. This fact motivates the development of layout tools which optimize layout area and chip performance simultaneously. This chapter focuses on the floorplanning step since it often plays a bigger role in overall circuit performance compared to the subsequent global and detailed routing phases.

8.2 Net-Based Approach

8.2.1 Overview

Many researchers have addressed the timing-driven placement. These research efforts are mainly tailored to layout styles which have regular structure such as gate array and standard cell design styles. [Prasitju 89] is the only work which presents a technique for the timing-driven placement of the general cells. [Jackson 87] has addressed the timing-driven global routing for the general cell layouts.

One common approach for solving the timing-driven placement problem is to transform timing constraints into net weights and to use these weights to guide the placement process [Dunlop 84, Burstein 85, Ogawa 86]. This step is repeated and weights are dynamically adjusted until all timing constraints are met. Another approach transforms timing constraints into maximum interconnection lengths [Nair 89, Ogawa 90] and then places the cells based on these net length constraints.

Timing requirements are often represented as path-delay constraints from the primary inputs or the outputs of the sequential logic blocks to the primary outputs or the inputs of other sequential blocks. There are two approaches to fill the gap between a net-based timing model and a path-delay input specification format. The first approach relies on a static timing analyzer which generates timing constraints based on the required arrival times at the primary inputs and outputs prior to the placement step [Nair 89]. The second approach uses a timing simulator incorporated into the placement process which dynamically adjusts timing constraints for each net [Dunlop 84, Ogawa 86, Teig 86].

8.2.2 The Procedure

A path is expressed as a sequence of nets between any source-sink pair in the circuit. Timing requirements for layout can be written as

$$\sum_{i=1}^{k} t(n_i) \leq S_p$$

where n_i is a net in the path, S_p is the slack of the path and $t(n_i)$ is the delay allowed to be used on net n_i. Slack for a path may be positive (indicating early arriving signal at the path end point) or negative (indicating late arriving signal). It is calculated as

$$S_p = T_{eff} - T_p^{max}$$
$$T_{eff} = T_{period} - T_{skew}^{max} - T_{setup} - T_{clk \rightarrow Q}$$
$$T_p^{max} = \sum_{n \in p}(\tau_n + R_{out,n}\ C_{fanout,n})$$

where τ_n is the intrinsic delay through node n, $R_{out,n}$ is the output resistance of n, $C_{fanout,n}$ is the fanout load seen by n, T_{period} is the clock cycle time, T_{skew}^{max} is the maximum clock skew, T_{setup} and $T_{clk \rightarrow Q}$ are the setup time and the internal clock to output delay for the synchronizing elements respectively.

The slack must be distributed to nets on the path [Hauge 87]. Normalized slack value is defined as $\hat{S}_p = S_p/k$ for a path with slack S_p and k nets. Let $C_{nominal}$ denote the nominal capacitance for all nets. Then, the capacitance constraint for net n_i is given by

$$C_{bound} = C_{nominal} + \frac{Min_{p \in paths(i)} \hat{S}_p}{R_{out,i}}$$

where $paths(i)$ denotes the signal paths which go through source i.

The capacitance constraint on delay given for each net is converted to a wire length constraint for the net. This is because the net length constraint is more easily handled during floorplanning. This conversion is based on a simple delay model which assumes a lumped capacitance at each input pin and a linear on-resistance for each driving pin. The upper bound on the net length, $length_{bound}$ is calculated as follows

$$length_{bound} = \frac{C_{bound}}{C_{unit}}$$

where C_{unit} is the wiring capacitance per unit length.

To make the delay calculation procedure less pessimistic, one must differentiate delays between rising and falling signals. Distinct values are provided for each signal type, and unateness information about each output pin of each macro is used during the path tracing, so that the proper values of the intrinsic and extrinsic delay are selected.

It is also possible that the designer has specified the delay constraints as net delays in which case the net wire length constraints are easily calculated using the output resistance of the driving cell and C_{unit}.

The multi-way cluster tree is generated based on block connectivities and net length constraints. Assume that root of the cluster tree is at level 0 and the leaves are at level d. A net is *l-critical* if it is a net with length constraint which must be handled at level $\geq l$ of the cluster tree. Otherwise, with high probability the net length constraint associated with this net can not be satisfied in the subsequent steps.

Given the net list, a complete graph $G(V, E)$ is built where each vertex represents a leaf cell and each edge corresponds to connections between pairs of cells. Initially, edges are assigned weights according to the number of connections between end points of the edge (i.e., their "natural connectivities"). A set of *d-critical* nets are identified and graph edge weights are updated to force merging of leaf nodes which are connected by these nets. Then, the matching algorithm described in Subsection 7.2.2 is used to form the clusters. This process is recursively applied until a rooted tree is constructed.

A key question is how to find a *minimal set of level critical nets* (MSLN) at level l of the tree. At this time, levels d up to $l+1$ of the tree have been generated, and hence, the interconnection length which is necessary to build the partially completed connection tree for each critical net can be easily calculated. This length is denoted by $length_{bu}$ of the net. The expected length of a critical net on p pins $length_{cur}$ at level l of the tree is estimated as (Figure 8.1)

$$length_{cur} = \rho(p) \times 2 \times \sqrt{A_a + A_w}$$

where $\rho(p)$ is ratio of the maximum length of a minimal Steiner tree (MST) connecting p pins on the net to the half perimeter length of the bounding box enclosing the net pins [Chung 79], A_a and A_w represent sum of the active and the routing areas of *strongly connected* clusters at level l. The notion of strongly connected clusters is related to that of the net neighborhoods which will be defined in Section 9.3. These are clusters which are at *distance* zero or one from the net and are connected to one another by edges with weights greater than a threshold. This threshold is the minimum weight on all the edges for the net in question.

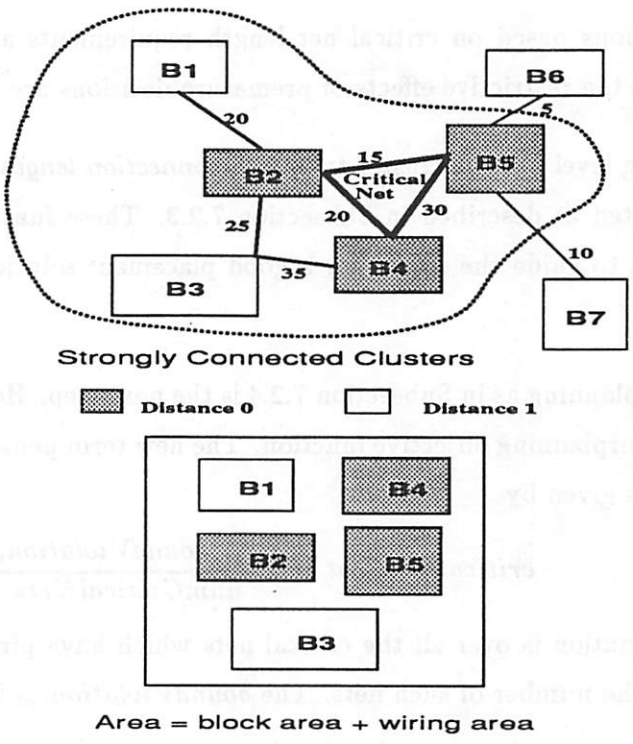


Figure 8.1: Net neighborhood population for a macrocells

An *l-critical* net is one which satisfies

$$length_{bound} - length_{bu} - length_{cur} \leq T_\epsilon$$

where $T_\epsilon > 0$ is the *criticality threshold.* After computing MSLN, the graph edge weights are updated to force merging of the tree nodes connected by MSLN. The updated weight for an edge belonging to MSLN, w_{new} is given by

$$length_{tot} = length_{bu} + length_{cur}$$

$$w_{new} = bias + w_{old} + \sum(\frac{length_{bound} - length_{tot}}{length_{bound}})^2$$

where w_{old} is the weight before updating and *bias* is the maximum value of the old connectivities. The *bias* is set to some large value in order to force merging of the nodes.

Reasons for choosing a *minimal* set of level critical nets are the following: Satisfying net length constraints may increase the overall cost of the cluster tree (in terms of the total interconnection length and layout area). In general, the earlier a commitment is made toward satisfying a particular net length constraint, the larger this increase is. Hence, the

clustering decisions based on critical net length requirements are postponed as much as possible so that the restrictive effects of premature decisions are avoided.

After generating level l of the cluster tree, *interconnection length function* s for the critical nets are computed as described in Subsection 7.2.3. These functions are used during the top-down phase to guide the search for a good placement solution while satisfying timing constraints.

Top-down floorplanning as in Subsection 7.2.4 is the next step. However, a new cost term is added to the floorplanning objective function. The new term penalizes net length constraint violations and is given by

$$criticalNetCost = \frac{\sum_{nets} boundViolation_{net}}{numCriticalNets}$$

where the summation is over all the critical nets which have pins within the node. *numCriticalNets* is the number of such nets. The $boundViolation_{net}$ is calculated as

$$boundViolation_{net} = \begin{cases} (\frac{length_{tot}}{length_{bound}})^2 & \text{if } length_{tot} > length_{bound} \\ \frac{length_{tot}}{length_{bound}} & \text{otherwise} \end{cases}$$

$$length_{tot} = length_{intra} + length_{inter} + length_{td}$$

where $length_{intra}$ is the length of wires used within subclusters of the cluster node in question, $length_{inter}$ is the length of wires required to complete connections among subclusters and to the current I/O pins for the cluster node and $length_{td}$ is the length of wires *used-up* in order to produce the partial placement solution starting from the root. Expressions for $length_{intra}$ and $length_{inter}$ were given previously; $length_{td}$ is easily calculated by adding the $length_{inter}$ of the already placed cluster nodes.

8.3 Path-Based Approach

8.3.1 Overview

Prasitjutrakul et al. [Prasitju 89] presented a mathematical programming approach for the timing-driven initial placement of macro-cells. The authors used a source-sink connection structure to model the interconnections among various cells. They assumed that the delays

of wires connected to the same signal net are independent. However, delays through wires connected to the same net are closely related and an independent assignment of delays to these wires does not produce a valid timing model. They adopted the total normalized interconnection delay for signal paths as their objective function. This function does not, however, consider the minimization of total interconnection length. A timing-driven placement algorithm which minimizes both of these metrics while satisfying the timing and geometric constraints is needed. Furthermore, the number of timing constraints is proportional to the number of signal paths which could be exponential and the resulting non-linear programming problem is non-convex and can be solved only by breaking it into separate steps. An approach that gives rise to fewer number of timing constraints is more desirable. [Jackson 90, Srinivasan 91] have presented such an approach.

8.3.2 The Procedure

Modules are modeled as circles whose radii are equal to the square root of the estimated or actual area of the module [Hsh 87, Alon 88, Prasitju 89]. This model is used since the actual shapes of the modules are not known and also because it leads to more elegant and simpler mathematical formulation. A source-to-sink net model is assumed, that is, connections are modeled only between modules having a source-to-sink relationship [Jackson 89, Prasitju 89, Jackson 90, Srinivasan 91].

Let M, P and N denote the number of modules, pins and nets in the circuit, S and E refer to the set of path starting points (primary-inputs or synchronizing elements) and path end points (primary-outputs or synchronizing elements) respectively, T_s and T_e represent the actual arrival times at the path starting points and the required arrival times at the path end points, p_i denote a pin in the circuit, a_i refer to the actual arrival time at pin p_i, m_i be a module with radius r_i and center coordinates (x_i, y_i), and A represent the set of edges in the circuit which capture the source-sink relationships.

The objective of the timing-driven initial placement is to find a set of module positions which yields the minimum total interconnection length (using Euclidean Squares distances) subject to timing and geometric constraints. The timing constraints ensure that the circuit satisfies required time constraints at the end points of all signal paths. The geometric

constraints ensure that modules do not overlap. This is mathematically described as:

$$
\begin{aligned}
&\textit{Minimize} \quad L = 1/2(w^T Q w + b^T w) \\
&\qquad \textit{subject to} \\
&a_j \geq a_i + \tau(p_i, p_j) && \forall (p_i, p_j) \in A \\
&a_i \geq T_s && \forall p_i \in S \\
&a_j \leq T_e && \forall p_j \in T \\
&d(m_i, m_j) \geq r_i + r_j && \forall m_i, m_j \in M, m_i \neq m_j.
\end{aligned}
$$

w is the vector of $2M + P$ coordinate and arrival time variables, Q is the $(2M + P)(2M + P)$ matrix of the cost function, b denotes contributions from fixed modules (I/O pads), $\tau(p_i, p_j)$ is the propagation delay from p_i to p_j[1] and $d(m_i, m_j)$ is the Euclidean distance between the center points of modules m_i and m_j.

This formulation is adopted from [Srinivasan 91] with the addition of geometric constraints. Without the geometric constraints, the formulation is a convex programming problem (convex objective function and convex constraint set)[2], and hence, efficient numerical techniques can be applied to it (P-I). Unfortunately, as the result of adding the geometric constraints, the constraint set becomes non-convex (P-II).

In one approach, P-I is solved to obtain a point placement. Next, either first a slicing tree is derived from the point placement followed by rectangle dissection to find the optimal shapes of modules as in [Otten 82, Otten 83] or the slicing tree and the optimal shapes of modules are simultaneously computed from the point placement as in [van Ginneken 90]. The difficulty with this approach is that the overlap among various modules may be excessive and deriving a reasonable slicing tree based on this type of point placement becomes questionable.

[1] If p_i and p_j belong to the same module, τ denotes the sum of the intrinsic (due to propagation delay through module) and extrinsic (due to input pin capacitances of the modules driven by p_j) delays. Otherwise, it denotes the propagation delay through interconnecting wires, in which case it is given by

$$\tau(v_i, v_j) = R_i \ [C_H|x_i - x_j| + C_V|y_i - y_j|]$$

where R_i is the output resistance of pin p_i and C_h (C_V) denotes the horizontal (vertical) capacitance per unit length of horizontal (vertical) interconnect wires.

[2] Although the equation for τ is non-linear, a mathematical transformation can be used to convert it to an equivalent linear equation at the expense of introducing $4N$ new variables. However, these new variables do not enter the cost function, and therefore, sparsity of the Q matrix is maintained. (See [Srinivasan 91].)

P-II may be solved iteratively, that is, first P-I is solved using techniques presented in [Jackson 90, Srinivasan 91]. Consequently, a timing feasible initial placement with module overlaps is obtained. Next, overlaps are identified and appropriate geometric constraints are added to P-I in order to eliminate overlaps. For example, if modules m_i and m_j overlap, the following two constraints will be added

$$|x_i - x_j| \geq r_i + r_j$$
$$|y_i - y_j| \geq r_i + r_j.$$

Sub-problem P-I^1 which is a non-convex non-linear programming problem is formulated and solved using the Penalty method [Fletcher 81]. To avoid numerical unstabilities, the dual solution must be considered during the optimization process by applying the SQP-method [Schittkowski]. The process may be repeated until a solution to II is obtained. Global spacing, global routing, shape optimization procedure given in Section 7.4 and pin assignment procedure given in Section 7.3 can subsequently be used to produce a feasible and area optimized solution.

In fact, one need not obtain a legal solution to II, that is, the iteration may be stopped at step k (after solving sub-problem P-I^k) when some modules possibly overlap, but they are generally distributed over the layout plane. Then, one of the following options may be used: 1) Eliminate overlaps and do shape optimization either by adding topological constraints to an incomplete constraint set to obtain a complete extension of the set [Dong 89], or by starting with a complete constraint set that may have redundant constraints and removing those redundant constraints that result in a reduction of floorplan area [Vijayan 90]; 2) Start with modules as points and gradually expand modules to meet their specified area and shapes as in [Yonezawa 90]; 3) Construct either a multi-way cluster tree or a binary slicing tree based on the coordinates of the modules and then floorplan the tree using procedures presented in Sections 7.2 and 7.3. As mentioned earlier, the advantage of using a cluster tree over a binary slicing tree (which has two-way nodes with prespecified cut orientations) is that it gives the maximum flexibility to the floorplan procedure and allows exploration of more topological possibilities.

8.4 Experimental Results

The net-based approach has been implemented and incorporated into BEAR-FP. I carried out experiments on the *MCNC* benchmarks in order to evaluate the performance of the net-based floorplanning procedure. Tables 8.1 and 8.2 depict the critical net lengths for *Xerox* and *Ami33*. (Chip aspect ratios are set to 2.0.) The first column gives the net name, the second column specifies the upper bound lengths, the third and fourth columns correspond to the estimated interconnection lengths after routing with conventional placement and with timing-driven placement. The upper bound length constraints for *Xerox* have been specified as part of the *MCNC* benchmark set. For *Ami33*, I assigned tight upper bound constraints to eight of the nets.

net name	upper bound	conventional	timing-driven
A1PC	850	281	220
NRLFTA	600	5432	560
TXREQ	600	906	526
PAORB	1200	671	873
A1A	2000	5648	1294
LK	850	684	731
ID	2200	3453	1720
RXOPDA	850	254	206

Table 8.1: Critical net length results for *Xerox* circuit (all values are in μ meters)

net name	upper bound	conventional	timing-driven
7	600	1859	299
99	600	488	338
222	700	2787	77
257	700	2215	645

Table 8.2: Critical net length results for *Ami33* circuit (all values are in μ meters)

Timing-driven placement of *Xerox* produces a layout with 6.5% increase in chip area and 3.1% increase in total interconnection length (compared to placement without length constraints). Timing-driven placement of *Ami33* produces a layout with 2.1% increase in chip area and 1.7% increase in total interconnection length. (See Table 8.3.) This is to be expected since satisfying the critical net length constraints often leads to an increase in chip area and/or total interconnection length. There are no upper bound length violations for the benchmark examples.

example	placement	chip area	total wire length
Xerox	conventional	25.6	613.8
	timing-driven	27.3	632.3
Ami33	conventional	2.69	132.3
	timing-driven	2.75	134.5

Table 8.3: Chip area (mm^2) and total interconnection length (mm) results for *Xerox* and *Ami33* circuits

Chapter 9

Area Estimation

9.1 Introduction

9.1.1 Motivation

Interconnection analysis addresses two related problems: the wire (interconnection) length and distribution problem and the wiring area estimation problem. Many researchers have addressed the latter problem and good area estimation techniques are available. However, the wire length and distribution problem has not been solved satisfactorily. Interconnection length studies tend to be theoretical and hence not applicable to specific designs, empirical such as those relating Rent's rule parameters to the average wire length, or based on experience with previous design layouts. Early research into interconnection length estimation, although of theoretical interest, is too general to be useful for specific designs. Later work, which produces results that have the appropriate level of detail, requires knowledge of *Rent's exponent* or assumes particular wire length distributions. In practice assumptions about wire length distributions are either not verified or require fitting curves to the actual layout data.

A method which describes the wire length distribution as a function of the specific logic design before layout is needed. Good physical design of large systems requires accurate area estimates of the individual modules for area planning, optimal placement, and routability

predictions. This requirement encourages development of procedural models which include characteristics of the physical design processes, structural description of the logic design, and physical features of primitive cells. These models produce interconnection length estimates with high accuracy (without making arbitrary wire length distribution assumptions or fitting curves to the data).

Interconnection length models have many uses. They can evaluate the capability of a new fabrication technology. The models determine routability of the proposed logic design, subject to the constraints of the technology, and therefore, help the system designers trade off aspects of the design and the technology. Accurate interconnection length estimates are beneficial since they measure the quality of placement and global routing algorithms. Interconnection estimates are also useful during the technology mapping phase of the logic synthesis since they can predict the cost of various implementations.

The area required for interconnections within a circuit layout largely depends on the total length of wire that must be accommodated. Therefore, accurate estimation of total interconnection length is an essential part of any area estimation procedure. In fact, many area estimation techniques require the wire length for the logic design as an input parameter [El Gamal 81a, Kurdahi 89].

9.1.2 Prior Work

Interconnection analysis models are divided into three categories: empirical, theoretical, and procedural. Empirical studies produce expressions for physical characteristics by extracting information from actual designs and fitting curves to the data. Theoretical studies produce closed form expressions by making simplifying assumptions about the interconnection structure. Procedural models consider more detailed aspects of the actual design processes, physical structures and interconnection structure of the design to improve the accuracy of the predictions.[1]

[1] For a more comprehensive review, refer to [Hanson 88].

Empirical Models

The initial work on the wiring requirements was performed by Rent in early 1960's. He derived *Rent's rule* which is a relationship between the I/O count and the cell count of a design by fitting curves to the empirical data from various computer designs

$$ioCount = (averageCellSize) \times (cellCount)^r$$

where r is *Rent's exponent.* Landman and Russo [Landman 71] studied the relation between cell versus I/O counts and the Rent's exponent. They showed two different values of Rent's exponent must be used depending on the number of cells, that is, the circuits with larger cell counts and smaller package counts have smaller Rent's exponents. Donath [Donath 79] reported that values of Rent's exponent ranged as high as 0.75 for highly parallel designs and as low as 0.47 for highly serialized designs. Sastry and Parker [Sastry 86] derived an interconnection length distribution that fitted actual designs.

These models require knowledge of empirical parameters (such as Rent's exponent) that are computed from actual design instances. An implicit assumption is that the design instances used in deriving the values of these parameters have the same interconnection structure and design characteristics as those of the design under consideration. This assumption limits the applicability of the empirical formulas.

Theoretical Models

Theoretical models produce closed form, mathematical descriptions of the physical characteristics from logic designs and physical implementation technologies. These models provide general trends but lack sufficient detail to represent individual designs accurately. They are useful when little is known about the actual design process. These models are divided into two categories: deterministic and stochastic.

Deterministic models rely on parameters extracted from actual design instances. The effects of the physical design processes are characterized by simple, measurable parameters. Donath [Donath 69] devised a plausible structure for a logic design which conforms to Rent's rule. He assumed a hierarchical structure where only a fraction of the pins inside a cell are

connected to pins outside the cell (the "encoding" assumption). He showed that such a structure exhibits Rent's rule. He also demonstrated that a randomly constructed design does not conform to Rent's rule.

A major thrust in stochastic approaches models the interconnection characteristics of the design as a stationary process. The wiring requirements are computed by making assumptions about the probability distributions of wires. An early attempt to formalize the characteristics of computer logic designs was published by Donath [Donath 70]. He defined a top-down hierarchical design approach in which each step of the expansion of the hierarchy is modeled by the substitution of a pattern of interconnected cells for each block. These patterns are selected randomly from a fixed pattern library by a stochastic process. Based on this model, Donath established the relation between the cell-to-pin ratio and performance.

Heller et al. [Heller 77] addressed the problem of estimating wiring space requirements. He modeled interconnection wires as independent two-point wires originating stochastically (with a Poisson distribution) at some cell, covering a random distance (an average interconnection length) and terminating at some second cell. Based on this model, he derived the probability of wiring completion of some number of cells in a limited number of wiring tracks. His model correctly predicts the relative difficulty of wiring completion in various designs. El Gamal [El Gamal 81a] refined Heller's model. His model assumes a regular two-dimensional array of cells. The generation and length of interconnecting wires are modeled as in Heller's work. The path traveled by each wire is established randomly, with the restriction that its endpoints be separated by a Manhattan distance which is equal to the path length. El Gamal derived from this model the minimum number of wire segments, and hence the minimum wiring area required for the square array of cells. He concluded that the overall minimum wiring area is of order $N^2 log^2 N$ where all cells have been placed in an $N \times N$ array.

Sastry and Parker [Sastry 86] used a model similar to El Gamal's. They modeled interconnections as independent two-point wires covering an average length and derived expressions for channel widths, probability of routing completion, and wire lengths. They showed that wire length distribution has the form of a Weibull distribution with location and shape parameters. These parameters must be computed based on the net lengths obtained from actual layouts. Kurdahi and Parker [Kurdahi 89] presented an area estimator for standard

cell layouts. They assumed rows of equal size, double entry cells, constant pin pitch, two-pin nets and minimal rectilinear connection paths. Their model assumes *birth* of a wire at pin slot i and length of a wire l are independent random variables with probabilities $p_B(i)$ and $p_L(l)$. They suggested uniform distribution for $p_B(i)$ and geometric distribution for $p_L(l)$. Based on these assumptions, the required routing area is estimated. This model, however, requires knowledge of average interconnection length which is computed by fitting curves to known data.

These models, although of great theoretical interest, are too general to be useful for specific design decisions. They require knowledge of empirical parameters or hypothetical wire length distributions. Assumptions about wire length distributions are either not verified in practice or require fitting curves to the actual layout data. Many area estimators require wire lengths as input. The accuracy of the area estimates is, therefore, bounded by the accuracy of interconnection length estimates. To be useful for design work, however, estimates with 10% accuracy are needed. In order to achieve this level of accuracy, proper abstractions to model layout processes and physical structures as well as careful analysis of the interconnection structure of the design under consideration are necessary. Theoretical models lack this level of detail and therefore produce results that are not accurate enough for today's design work.

Procedural Models

Procedural models incorporate greater detail and a lower level of abstraction compared to other models. They rely on relations derived from knowledge of the actual design processes, interconnection structure of the design, physical layouts of the leaf cells and layout rules. These models extract interconnection characteristics of the design and combine them with abstractions of the placement and routing processes to give estimates without need for arbitrary wire length distribution assumptions or empirical parameters.

Sechen [Sechen 87b] presented an interconnection length estimator which gives accurate estimates for small designs. He assumed square cells placed on an square, two-dimensional grid. For each size of net, the half perimeter of the smallest rectangle enclosing all pins on the net is computed. Various scenarios and a look up table are used to determine all possible

arrangements of cells which establish a given bounding box. Total interconnection length is then computed by summing (over all nets) the half perimeter lengths of the rectangles enclosing pins on the nets. Sechen's abstraction of the layout surface make his model most applicable to "sea-of-gates" style. His approximation of total interconnection length for nets with large number of pins (> 4) is not accurate enough.[2] I implemented an interconnection length estimator based on half perimeter lengths of net bounding boxes. For the circuits in the test suite, errors up to 30-40% were observed.

Chen and Bushnell [Chen 88] introduced an area estimator for random placement with the assumption that wires do not share tracks. They derived the expected number of wiring tracks, and feedthroughs in the central row, and thereby, estimate the chip width and height. The authors do not attempt to model global and detailed routing processes, and do not differentiate between designs based on their interconnection structures. Their estimated chip area for small designs is 40-70% over the actual chip area, and the number of wiring tracks is overestimated by a factor of 2-3. No data is presented for medium or large size designs.

9.1.3 Overview

To obtain accurate area estimates, it is necessary to achieve high accuracy in estimating the wire length. This task is accomplished by the *procedural* model presented here. The model captures the characteristics of the physical design processes (placement, global routing and detailed routing) and the structural features of the logic design in order to accurately estimate interconnection length.

This model produces accurate interconnection length estimates for standard cell layouts. Since interconnection length is a very strong function of a logic design, the first task is to extract relevant features which account for the wiring requirements of the logic design. A metric which captures the local influence of other nets over a net under consideration is introduced. This is a more pertinent and effective metric (as far as interconnection length estimation is concerned) compared to other metrics such as average pin per cell or average

[2] Chung [Chung 79] showed that the worst case length of a minimal rectilinear Steiner tree connecting d pins of a net tends to be $(\sqrt{d}+1)/2$ of the half perimeter length of the smallest rectangle enclosing pins of that net.

number of connected cells to any cell in the design.

The wire length model relies on knowledge of the actual design processes (placement, global routing and detailed routing), and physical structures. The predicted results are obtained from analysis of the net list. No prior knowledge of the functionality of the design is used. The model considers multi-pin nets directly, and does not preprocess them into sets of 2 pin nets (as is often the case). Using these wire length estimates, the chip width and height can be computed by statistical area estimation presented in Section 9.2 or by random offset track packing technique introduced in Section 9.3.

Two interconnection models are presented [Pedram 89a, Pedram 89b]. The *basic* model features a random placement but optimized global and detailed routing. Since the random placement process can be characterized accurately, the effects of placement and routing within the overall model can be separated. The *improved* model extends the basic model by including optimized placement and is used in production. Optimized global and detailed routing abstractions from the basic model are retained.

Given knowledge of standard cell layouts and model assumptions, the model equations follow logically (without reference to any empirical or arbitrary parameters).

9.2 The Basic Interconnection Model

The inputs to the area estimation model are the logical design specification and primitive cells included in the specification. Following the standard cell model, double entry cells are placed in rows and interconnected in routing channels among the rows. Outputs of the estimation model are the estimated total wire length, wire length distribution, the estimated total number of feedthroughs, the feedthrough distribution, chip width and height and chip area.

A standard cell layout is modeled as a regular $w \times n$ array, where n is the number of rows and w ($= numCells/n$) is the average number of cells per row. Wires follow rectilinear paths with horizontal segments on one layer (called *metal1* or $M1$) and vertical segments on another (called *metal2* or $M2$). The average cell width is computed from the cells actually used in the design.

Pins on a net are *distinguishable*, that is the number of states available to a net of N pins is the product of the number of states available to each pin. Notice the implicit assumption here that the states available to any one pin do not depend on whether they are already occupied by other pins.

The basic model assumes a random placement but optimized global and detailed routing processes. The following important aspects of the algorithms have been incorporated. The placement process uniformly distributes cells on the $w \times n$ grid. The global router finds a minimum spanning tree to connect pins of nets. Wiring for a net does not meander outside the bounding box defined by the pins on the net. Feedthroughs are placed at the intersections of cell rows and the edges in the spanning tree connecting pins on the net. No feedthrough is added to a row which contains a pin on the net. Each net contributes at most one feedthrough to each cell row. A channel routing paradigm is assumed. The channel router finds the shortest path inside the channel to connect pins on the net. The route does not meander outside the box enclosing these pins. Inside the channel each net is connected with trunks with no overlap along the length of the channel. Branches connect trunks to the pins. All branch layer conflicts can be resolved by adding horizontal jogs. Over-the-cell routing is not considered.

The assumption of independent nets allows us to compute the wire length and feedthrough contribution of each net separately. The random placement assumption implies uniform pin distribution over the layout surface and is captured in the $M1LengthInFrame$ and $M2LengthInFrame$ equations. Consider a net with d pins uniformly distributed on a $w \times n$ frame. Sum of the lengths of *metal1* wires connecting all pins on the net (in units of cell pitch) is computed as

$$M1LengthInFrame(d, w, n) = \sum_{i=1}^{Min(d,n)} (\frac{1}{n})^d \times \begin{pmatrix} n \\ i \end{pmatrix} \times A(i, d, w).$$

The first term gives the probability of placing d pins on some subset of n rows. The next term gives the number of ways i rows can be selected from among n rows, and $A(i, d, w)$ gives the contribution of a d-pin net occupying exactly i rows ($i \leq d$) to the *metal1* length.

In order to compute $A(i, d, w)$, all different configurations (groupings) of d pins on i rows

are examined. In particular, the following integer equation must be solved

$$\sum_{j=1}^{i} x_j = d \qquad 1 \leq x_j \leq w.$$

The solution to this equation returns a list of *sets*. Each set represents a *distinct* pin configuration describing the distribution of pins on rows. For example, if $i = 3$, $d = 6$, $w = 60$, then solution to the integer equation is $((1,1,4),(1,2,3),(2,2,2))$. Elements in each set are increasing in magnitude, that is, $(1,1,4)$ is an acceptable pin set, but $(1,4,1)$ is not. This equation is efficiently solved by a recursive procedure. The cardinality of the solution (list of sets) strongly affects the run time of the model since the number of solutions grows rapidly with d and i. Therefore, results for very large nets are approximated by dividing large nets into cliques of smaller nets. Now,

$$A(i,d,w) = \sum_{sets} CfgLength(i,w,set) \times A_1(i,d,set) \times A_2(i,d,set)$$

$$A_1(i,d,set) = \prod_{k=1}^{d-1} \begin{pmatrix} i - \sum_{j=1}^{k} rows[j] \\ rows[k+1] \end{pmatrix} = \frac{i!}{\prod_{k=1}^{d} rows[k]!}$$

$$A_2(i,d,set) = \prod_{k=1}^{i-1} \begin{pmatrix} d - \sum_{j=1}^{k} pins[j] \\ pins[k+1] \end{pmatrix} = \frac{d!}{\prod_{k=1}^{i} pins[k]!}$$

where $A_1(i,d,set)$ is the number of distinguishable row arrangements for a given pin set and $A_2(i,d,set)$ is the number of distinguishable pin distributions for a given row arrangement and a given pin set. $rows[k]$ is the number of rows with k pins, and $pins[k]$ is the number of pins on the kth row. For example, if $set = (1,1,4)$, then $rows[1] = 2$ and $pins[1] = 1$. For this pin set, distinguishable row arrangements are $(1,1,4)$, $(1,4,1)$ and $(1,1,4)$. For row arrangement $(1,1,4)$ and assuming pins are numbered as $p1,\cdots,p6$, then $((p1),(p2),(p3,p4,p5,p6))$, $((p2),(p1),(p3,p4,p5,p6))$, $((p2),(p3),(p1,p4,p5,p6))$ and so forth are distinguishable pin distributions.

$CfgLength(i,w,set)$ which gives the expected length on the net if it assumes the distribution of pins described by a particular set is computed next. $CfgLength$ is an abstraction of the global router, assumes that pins (on a net) on rows do not share channels with pins on other rows and is given by

$$CfgLength(i,w,set) = \begin{cases} WL(pins[1],w) & if \text{ i } = 1 \\ \sum_{k=2}^{i} WL(pins[k]+1,w) & else\ if \text{ pins[1] } = 1 \\ WL(pins[1],w) + \sum_{k=2}^{i} WL(pins[k]+1,w) & otherwise \end{cases}$$

where $pins[k]$ is sorted in increasing order.[3] $WL(m,w)$ gives the expected length of the net which has m pins in a routing channel ($2 \leq m \leq 2w$) and is calculated as

$$WL(m,w) = \frac{\sum_{l=m}^{w} 4(w-l+1) \times \begin{pmatrix} 2l-2 \\ m-2 \end{pmatrix} \times l}{\begin{pmatrix} 2w \\ m \end{pmatrix}}.$$

The numerator is a sum over all possible spans of the m randomly placed pins on a channel with w cells on each side and the denominator is the number of ways m cells can be chosen from among $2w$ cells. The first term in the numerator is the number of ways spans of l cell pitches can be established within the channel, the second term in the numerator is the number of ways the remaining $m-2$ pins can be placed on $2l-2$ cells, and l is the cell span established by the pins.

Under the assumption of a single wire segment per track (using unity track demand factor, i.e., $\forall m\ WL(m,w) = 1$), the equation for $M1LengthInFrame(d,w,n)$ reduces to that of [Chen 88]

$$A(i,d,w) = i \times \sum_{sets} A_1(i,d,set) \times A_2(i,d,set) = i \times B(i,d)$$

$$B(i,d) = i^d - (\sum_{j=1}^{i-1} \begin{pmatrix} i \\ j \end{pmatrix} \times B(j,d)) \qquad B(1,d) = 1.$$

$B(i,d)$ which is defined recursively gives the number of ways of placing d pins on exactly i rows. Note that

$$\sum_{i=1}^{Min(d,n)} (\frac{1}{n})^d \times \begin{pmatrix} n \\ i \end{pmatrix} \times B(i,d) \equiv 1.$$

Sum of the lengths of *metal2* wires connecting all pins of the net (in units of channel height) is computed as

$$M2LengthInFrame(d,n) = \sum_{i=1}^{Min(d,n)} (\frac{1}{n})^d \times \begin{pmatrix} n \\ i \end{pmatrix} \times ChanSpan(i,n) \times B(i,d)$$

[3] These equations model a minimum spanning tree global router. A similar but different set of equations is used to model minimum Steiner tree global routers. Same comment applies to the equation computing $NumFTs$.

$$ChanSpan(i,n) = \frac{\sum_{l=i}^{n}(n-l+1) \times \begin{pmatrix} l-2 \\ i-2 \end{pmatrix} \times (l-1)}{\begin{pmatrix} n \\ i \end{pmatrix}}$$

where $ChanSpan(i,n)$ is the expected number of channels spanned by a d-pin net (occupying i rows). The first term in the numerator is the number of ways spans of l rows can be established within the chip, the second term in the numerator is the number of ways the remaining $i-2$ rows can be chosen from among $n-2$ rows, and $l-1$ is the channel span. The denominator is the number of ways i rows can be chosen from among n rows.

The expected number of feedthroughs contributed by a d-pin net is computed next

$$FTHeightInFrame(d,n) = \sum_{i=1}^{Min(d,n)} (\frac{1}{n})^d \times \begin{pmatrix} n \\ i \end{pmatrix} \times NumFTs(i,n) \times B(i,d)$$

where $NumFTs(i,n)$ is the expected number of feedthroughs added by a net which is occupying exactly i rows. It is given by an expression identical to that for $ChanSpan$ with $l-i$ (number of feedthroughs) replacing $l-1$ (channel span). This is because the global router does not add a feedthrough to a row which contains a pin on the net.

The total interconnection length required to connect all the nets and the total number of feedthroughs contributed by all the nets are

$$totM1Length = \sum_{nets} nets[d] \times M1LengthInFrame(d,w,n)$$

$$totM2Length = \sum_{nets} nets[d] \times M2LengthInFrame(d,n)$$

$$totFTs = \sum_{nets} nets[d] \times FTHeightInFrame(d,n)$$

where $nets[d]$ represents the number of nets with d pins. Note that the distributions of wire lengths and feedthroughs as a function of the number ($nets[d]$) or size (d) of nets in the logic design have been computed as well.

The abstraction of the channel routing process is composed of two components: the wire length abstraction captured by $WL(m,w)$ equation given previously and the segment packing into tracks abstraction described below.

The *metal1* length for each net is divided equally into a number of segments as determined by the expected channel span of the net.[4] The average segment length (over all nets) is computed as

$$segmentLength = \frac{totM1Length}{totSegments}$$

$$totSegments = \sum_{d} numSegments[d].$$

Because of random placement process, one may argue that the segments in the channel originate according to a Poisson distribution with parameter δ where δ is the average number of wire segments per slot and is given by

$$\delta = \frac{totSegments}{w \times (n-1)}.$$

Next, define

$$\alpha = \delta \times segmentLength = \frac{totM1Length}{w \times (n-1)}.$$

From this and a confidence level of c (= 0.999) for routing completion, the required number of wiring tracks per channel is approximated as follows

$$\sum_{k=1}^{tracksInChan} \frac{e^{-\alpha} \times \alpha^k}{k!} \leq c$$

$$totTracks = (n-1) \times tracksInChan.$$

After computing the total number of wiring tracks required by the detailed router, the number of feedthroughs crossing each of the rows is computed. The probability of a feedthrough crossing row i (rows are numbered from bottom to top starting from 1) is given by

$$PFTOnRow(i,n) = \sum_{j=1}^{d-1} (\frac{i-1}{n})^j \times (\frac{n-i}{n})^{d-j} \times \begin{pmatrix} d \\ j \end{pmatrix}.$$

From the d pins on the net, assume that j are placed in rows below the ith row and $d-j$ pins are placed in rows above the ith row. $(i-1)/n$ is, then, the probability that one pin is placed in rows below row i and $(n-i)/n$ is the probability that another pin is placed in rows above row i. Note that if at least one pin on the net is placed on row i, then $PFTOnRow(i) = 0$. This is consistent with the assumption that the global router does not add feedthroughs to a row which contains some pin on the net.

[4] A spanning tree global router tends to divide the *metal1* wire length of a net equally among the channels spanned by the net.

For randomly placed designs, the number of feedthroughs crossing the central row is the largest. To compute the probability that a d-pin net will contribute a feedthrough to the central row, $i = (n+1)/2$ is used. Now,

$$CFTsInFrame(d,n) = PFTOnRow(\frac{n+1}{2}, n) \times FTHeightInFrame(d,n)$$

$$cFTs = \sum_{nets} nets[d] \times CFTsInFrame(d,n).$$

The $chanHeight$ is computed as

$$chanHeight = \frac{totTracks \times trackSpacing}{n-1}$$

and finally, chip width, chip height, and actual *metal1* and *metal2* lengths (in μmeters) are computed as

$$cellPitch = cellWidth + cFTs \times ftWidth/w$$

$$chipWidth = w \times cellPitch$$

$$chipHeight = n \times cellHeight + (n-1) \times chanHeight$$

$$actualM1Length = totM1Length \times cellPitch$$

$$actualM2Length = totFTs \times cellHeight + totM2Length \times chanHeight.$$

9.3 The Improved Interconnection Model

The improved model assumes a placement optimization process, a minimal rectilinear Steiner tree global router and a left edge channel router. The features of the algorithm classes which are captured by the interconnection model are the following. The placement optimizer minimizes the sum over all the nets of the half perimeter length of the rectangle enclosing pins of each net. Pins inside the placement bounding box for the net are not optimized for that net.

The global router approximates a minimal rectilinear Steiner tree to connect pins on each net. This global routing paradigm [Cong 88] tends to produce minimal *metal2* routes at the expense of more *metal1* routing. (See [Lee 88] for an opposing global routing paradigm that produces maximal *metal2* routing.) A channel routing paradigm is assumed. (See Section 9.2.)

Interconnection length and feedthrough count for each size of net is estimated and then summed over all the nets. *Metal1* wire length is expressed in units of average cell pitch and *metal2* wire length is expressed in units of average channel height. The average interconnection lengths are computed by spatially restricting the possible positions of the pins on the net to a bounding box within the $w \times n$ grid. Considering feasible aspect ratios for this bounding box and various pin configurations within the box and averaging over all such states, the average interconnection lengths and feedthrough count for the net are computed. By summing over all nets, the total interconnection length and the total number of feedthroughs are computed. There is no explicit dependence on a particular cell library or fabrication technology for estimation of wire length. However, such information is required when the total interconnection length is used to estimate the chip width and height.

The interconnection structure of a design is characterized by net *neighborhood populations* (NPs) which account for the local influence of other nets over a net in question. The NP for a net is the number of primary input/outputs (I/Os) and cells which are at distance zero or one from the net. To compute the NP for a particular net, we find all the cells and I/Os connected by this net (i.e. at distance 0 from the net). Every other net which is connected to the cells is followed all the cells and primary I/Os which are at distance one from the net are visited. The NP for the net is the total number of *distinct* cells and I/Os encountered in this manner. In the NP computation, nets which connect more than 25% of the cells in the design are ignored. (These are typically power and clock nets that go everywhere.) This procedure is repeated for all nets of given size resulting in the average neighborhood population for each size of net. At the end, $NP[d]$ contains the average neighborhood population for nets with d pins.

The NP for a net reflects the conflicting demands on a placement optimizer that is attempting to optimally place the cells directly connected to the net. To clarify this notion, assume that the placement optimizer is seeking a placement of d cells connected by exactly one net. The optimizer will cluster these cells in a bounding box of minimum half perimeter length. However, in reality, it is not possible to place cells connected to each net in such a minimum length bounding box due to competition from other nets. The placement and routing of the cells directly connected to a net of size d (to a first approximation) is influenced by a cell and I/O population of size $NP[d]$.

The abstraction of the placement optimizer is described next. Consider a d-pin net with pins on a two-dimensional, $w \times n$ grid. The d pins on the net can be placed within an $x \times y$ bounding box where y ranges from k_1 to k_2 and $x = \lceil NP[d]/y \rceil$. k_1 is given by $\lfloor NP[d]/XSpan(d,n) \rfloor$ and k_2 is equal to the $\lceil YSpan(d,n) \rceil$ where $XSpan(d,n)$ and $YSpan(d,n)$ are the the expected cell span and the expected row span of the net if the net pins are randomly placed on the $w \times n$ grid. Due to the placement process which minimizes the half perimeter length of the rectangle enclosing all pins on the net, and due to conflicting demands of other nets, the d pins are uniformly distributed inside the $x \times y$ bounding box. Now,

$$W(x,y) = \frac{(w-x+1) \times (n-y+1)}{x \times cellWidth + (\gamma \times (y-1) + y) \times cellHeight}$$

where the numerator gives the count of all feasible subgrids of size $x \times y$ in a grid of size $w \times n$, and the denominator gives the half perimeter length of the $x \times y$ grid. γ is the ratio of the expected channel height to the cell height.

The average length of the net with d pins is given by

$$M1Length(d) = \frac{\sum_{y=k_1}^{k_2} W(x,y) \times M1LengthInFrame(d,x,y)}{\sum_{y=k_1}^{k_2} W(x,y)}$$

where $M1LengthInFrame(d,x,y)$ is the expected length of the net if it is restricted to $x \times y$ bounding box and is given in Section 9.2. Recall that this length depends on $CfgLength(i,x,set)$. Here, $CfgLength(i,x,set)$ is redefined so that it captures the effects of a minimal rectilinear Steiner tree global router. $CfgLength(i,x,set)$ gives the expected length of the net when the net assumes the configuration described by a particular pin set. In addition, one must address the *channel sharing* problem, i.e., given a particular pin configuration what is the probability of these pins are facing the same channel. This issue is important because pins on two adjacent rows can be connected within the shared channel. $Sharing(y,i,l)$ which gives the probability of pins which are placed on i out of y rows sharing exactly l channels ($l \leq i/2$) is computed by a recursive procedure. Then,

$$\begin{aligned} &CfgLength(i,x,set) = \\ &\sum_{r=0}^{i/2-1} \{ (\sum_{s=0}^{r} Sharing(y,i,s)) \times (WL(pins[2r+1],x) + WL(pins[2r+2],x)) + \\ &\quad (1 - \sum_{s=0}^{r} Sharing(y,i,s)) \times WL(pins[2r+1] + pins[2r+2],x) \} + \\ &\quad (i/2-1) \times WL(2,x) + (if\ IsOdd(i)\ then\ WL(pins[i]+1,x)\ else\ 0) \end{aligned}$$

$WL(m,x)$ gives the expected length of m-pin portion of the net when all m pins lie on one channel ($2 \leq m \leq 2x$) and is given in Section 9.2.

Sum of the lengths of *metal2* wires connecting all the pins, the expected number of feedthroughs added to all rows and to the central row by a net of size d are computed in a fashion similar to $M1Length$ calculation. For example,

$$FTHeight(d) = \frac{\sum_{y=k_1}^{k_2} W(x,y) \times FTHeightInFrame(d,y)}{\sum_{y=k_1}^{k_2} W(x,y)}$$

where $FTHeightInFrame(d,y)$ is given in Section 9.2. The total *metal1* and *metal2* lengths required to connect all nets, the total number of feedthroughs crossing all rows, $totFTs$, and those crossing the central row, $cFTs$, are also calculated as in Section 9.2.

Given average wire length and wire length distribution, known statistical area estimation techniques can be exploited to estimate the total chip area and aspect ratio [Heller 77, Kurdahi 89, Sastry 86]. Here, I shall describe a new technique based on *random offset track packing* to model the detailed routing process.

The *metal1* length for each net is divided equally into a number of segments as determined by the expected number of wire segments (trunks) for each size of net. The number and lengths of all segments for each size of net lying in each channel are given by

$$segmentsInChannel[d] = \lceil nets[d] \times Segments(d) \rceil / (n-1)$$

$$segmentLength[d] = \lceil M1Length[d] / Segments(d) \rceil$$

$$Segments(d) = \frac{\sum_{y=k_1}^{k_2} W(x,y) \times SegmentsInFrame(d,y)}{\sum_{y=k_1}^{k_2} W(x,y)}$$

$$SegmentsInFrame(d,y) = \sum_{i=1}^{Min(d,y)} (\frac{1}{y})^d \times \begin{pmatrix} y \\ i \end{pmatrix} \times B(i,d) \times \sum_{j=0}^{i/2} (i-j) \times Sharing(y,i,j).$$

Now, the *track packing* problem (in the absence of vertical and horizontal constraints) can be defined as follows: Given t segments which must be placed in tracks of equal length w and given that segment i requires l_i units of track length, the objective is to determine the minimum number of tracks needed to accommodate all segments. This is the well known bin packing problem and is NP-complete [Garey 79]. There exist many heuristics which obtain packings that use a "small" fraction of tracks more than the optimal packing.

The simplification made by assuming that no pin constraints exist on the wire segments causes underestimation of the routing area. By generating a uniformly distributed offset for each wire segment in the channel, this shortcoming is remedied. One could build a horizontal constraint graph for these randomly positioned wire segments. The assignment of tracks to wire segments corresponds to the proper coloring of this constraint graph (which is by construction an interval graph). In the absence of vertical constraints, there exist simple optimal algorithms for coloring the interval graphs. The task at hand, however, is much easier because only density of the channel must be computed and this can be accomplished by a simple plane sweep technique. The total density of the standard cell layout (*totDensity*) is the sum of channel densities over all channels. Ignoring vertical constraints in the area estimation model produces small errors because modern dogleg routers often route channels at density. Then,

$$chanHeight = totDensity \times trackSpacing/(n-1).$$

Finally, chip width, chip height, and actual *metal1* and *metal2* lengths (in μmeters) are computed as in Section 9.2.

9.4 Complexity Analysis

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C(k)	1	2	3	5	7	11	15	22	30	42	56	77	101	135	176

Table 9.1: $C(k)$ values for k ranging from 1 to 15

The complexity of the interconnection model presented above is

$$O(n \times \sum_{k=2}^{d_{max}} C(k))$$

where $C(k)$ is defined as

$$C(k) = \sum_{i=1}^{k} \|Sets(i,k)\|$$

Table 9.1 gives values of $C(k)$ for k ranging from 1 to 15.

The run time is relatively independent of the size of design but is strongly affected by the maximum size of net being considered (d_{max}). For this reason, nets with more than 15

pins are divided into cliques of smaller nets. This division introduces little error because, typically, there are few large nets. Each execution of the model requires 2-5 minutes for the example runs.

9.5 Experimental Results

The interconnection model has been implemented in the Cedar language running on Xerox Dorado Workstations (2-MIPS machines) and incorporated the model into the DATools system developed at Xerox PARC [Barth 88].

example	cells	I/Os	nets	pins
16b adder	144	52	177	546
SnprCtl	95	30	114	331
RSD	210	89	211	670
64b counter	478	130	585	1537
Primary1SC	750	73	903	2801
Primary2SC	2907	107	3029	8758

Table 9.2: Summary of the example circuits used for the area estimator

Table 9.2 describes the examples used to test the model's predictions. The counter and the adder are circuits synthesized by the DATools system when no performance requirements are imposed. The adders are simple ripple-carry designs, the counters are carry-look-ahead designs. The *RSD* is part of a Reed-Solomon error correction circuit, and the *SnprCtl* is part of a cache controller. Primary1SC and Primary2SC are the benchmarks from the physical design workshop [Preas 87]. The placement is obtained by TimberWolfSC version 4.1 [Sechen 87a]. The global and detailed routers are discussed in [Cong 88, Deutsch 76].

Table 9.3 compares the model's wire length, area and aspect ratio estimates with the actual results obtained after global and detailed routing of the circuits. The estimates are all within 10% accuracy. The detailed characteristics of the model were verified by collecting data and generating statistics for the actual interconnection length and feedthrough count for each size of net and comparing it with the model's estimated values.

Table 9.4 gives this comparison for Primary1SC. Detailed results for other examples and

aspect ratios are comparable. Sources of error include average behavior modeling (rather than worst case behavior modeling) and incomplete characterization of the physical design processes.

example	rows	predicted				actual			
		M1 length	M2 length	circuit area	aspect ratio	M1 length	M2 length	circuit area	aspect ratio
SnprCtl	5	24.1	20.1	0.81	1.11	22.1	17.5	0.76	1.04
16b adder	8	24.0	23.6	0.81	2.66	22.1	21.5	0.76	2.50
RSD	6	59.6	52.2	1.70	1.08	62.1	48.3	1.61	0.98
64b cntr	12	226.2	254.6	5.84	1.53	238.6	238.2	5.36	1.54
Primary1SC	22	714.0	545.5	27.1	1.36	782.7	491.2	26.9	1.47
Primary2SC	34	3958.2	3422.3	109	1.36	4300.1	3050.0	113	1.48

Table 9.3: Comparison of estimates versus the actual results of wire length (mm), area (mm^2) and aspect ratio

pins	estimated		actual	
	M1 length	ft count	M1 length	ft count
2	484	0.353	506	0.484
3	843	0.797	835	0.597
4	1073	1.026	1100	0.530
5	1417	1.114	1474	0.846
7	2028	1.106	2630	0.833
12	4280	0.852	5689	1.330

Table 9.4: Detailed comparison of *metal1* length and feedthrough count for various sizes of nets for Primary1SC with 14 rows

Chapter 10

Concluding Remarks

In the first part of this dissertation, I studied the effects of interconnect on circuit area and performance, presented appropriate models and computational procedures for estimating delay during synthesis, described algorithms for I/O pad assignment and placement of Boolean networks, and most of all, put forth techniques for coupling logic synthesis to placement and for maintaining simultaneous and interactive data representations in logic and layout domains. There is still much work to be done here. Extension of layout-driven approach to the synthesis of sequential networks and Field Programmable Gate Arrays, investigation of layout-driven techniques for logic resubstitution, simplification and redundancy removal, finding an efficient solution to the problem of technology mapping with the objective of minimizing area under delay constraints, and driving cell-based placement with the information embedded in the logic graph are a few of possible research directions. Integrating buffering and gate sizing with the actual placement procedure is another way to improve the circuit performance based on detailed layout information. Another important issue is that of estimating delay through combinational logic during the technology-independent phase of logic synthesis.

In the second part of this dissertation, I addressed the floorplanning problem and presented a technique based on the cluster tree generation, shape computation and floorplan optimization which unifies many of the physical design steps (placement, shape calculation, pin assignment, global routing, and global spacing) in a hierarchical manner. More work needs

to be done in floorplanning for sea-of-gates design style and analog circuits, channel pin arrangement problem, and path-based performance-oriented floorplanning. The floorplanner can be used in the feedback loop of a high-level synthesis system, in which case constructive floorplanning techniques must be incorporated in the existing framework.

Early estimation of macrocell areas (from a complete or partial net list specification) is also important to a floorplanning system. The work presented in this dissertation for predicting the area of standard cell assemblies can be extended to other design styles (sea-of-gates, PLAs, etc).

And on this note, I end my dissertation.

Bibliography

[Abouzeid 90] P. Abouzeid, K. Sakouti, G. Saucier and F. Poirot, "Multilevel synthesis minimizing the routing factor," *Proc. 27th ACM/IEEE Design Automation Conf.*, pages 365-368, 1990.

[Aho 74] A. V. Aho, J.E. Hopcroft and J.D. Ullman, "The design and analysis of computer algorithms," *Addison-Wesley Publishing Company*, 1974.

[Aho 76] A. V. Aho and S. Johnson, "Optimal code generation for expression trees," *J. ACM*, pages 488-501, July 1976.

[Alon 88] A. Alon and U. Ascher, "Model and solution strategy for placement of rectangular blocks in the Euclidean plane," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-7, No. 3, pages 1062-1081, November 1988.

[Antreich 82] K. J. Antreich, F. M. Johannes and F. H. Kirsch, "A new approach for solving the placement problem using force models," *Proc. IEEE Int. Symp. on Circuits and Systems, Proc. ISCAS*, pages 481-486, 1982.

[Bakoglu 86] H. B. Bakoglu, "Circuits, interconnections, and packaging for VLSI," Addison-Wesley, 1990.

[Barth 88] R. Barth, L. Monier and B. Serlet, "Patchwork: layout from schematic notations," *Proc. 25th ACM/IEEE Design Automation Conf.*, pages 250-255, 1988.

[Brady 84] H.N. Brady, "An approach to topological pin assignment," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-3, pages 250-255, 1984.

[Brayton 82] R. K. Brayton and C. McMullen, "The decomposition and factorization of boolean expressions," *Proc. Int. Symp. Circuits and Systems*, Rome, May 1982.

[Brayton 87a] R. K. Brayton, "Algorithms for multilevel synthesis and optimization," G. De Micheli, A. Sangiovanni-Vincentelli and P. Antognetti, Editors, *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, Martinus Nijhoff, 1987.

[Brayton 87b] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang, "MIS: a multiple-level logic optimization system," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6, No. 6, pages 1062-1081, November 1987.

[Brayton 90] R. K. Brayton, G. D. Hachtel and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. the IEEE*, Vol. 78, No. 2, pages 264-300, February 1990.

[Breuer 77] M. A. Breuer, "Min-cut placement," *Design Automation and Fault-Tolerant Computing*, 2, October 1977.

[Burkhard 80] R.E. Burkhard and U. Derigs, "Assignment and matching problems: solution methods with Fortran programs," Springer Verlag, 1980.

[Burstein 83] M. Burstein, S.J. Hong and R. Pelavin, "Hierarchical VLSI layout: simultaneous placement and wiring of gate arrays," *Proc. VLSI*, pages 45-60, 1983.

[Burstein 85] M. Burstein and M. N. Youssef, "Timing influenced layout design," *Proc. 22nd ACM/IEEE Design Automation Conf.*, pages 124-130, 1985.

[Cai 90] Y. Cai and D. F. Wong, "An optimal channel pin assignment algorithm," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 10-13, 1990.

[Chen 83] N. P. Chen, C. P. Hsu, E.S. Kuh, C. C. Chen and M. Takahashi, "BBL: a building-block layout system for custom chip design," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 40-41, 1983.

[Chen 84] C. C. Chen and E. S. Kuh, "Automatic placement for building-block layout," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 90-92, 1984.

[Chen 86] H. H. Chen and E. S. Kuh, "Glitter: A gridless variable-width channel router," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-5, pages 459-465, 1986.

[Chen 87] H. H. Chen, "Routing L-shaped channels in non-slicing structure placement," *Proc. 24th ACM/IEEE Design Automation Conf.*, pages 152-158, 1987.

[Chen 88] N. P. Chen, "Building block routing - a symbolic approach," *Proc. IEEE Custom Integrated Circuits Conference*, pages 11.2.1-11.2.4, 1988.

[Chen 88] X. Chen and M.L. Bushnell, "A module area estimator for VLSI layout," *Proc. 25th ACM/IEEE Design Automation Conf.*, pages 54-59, 1988.

[Cheng 84] C.-K. Cheng and E.S. Kuh, "Module placement based on resistive network optimization," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-3, pages 218-225, July 1984.

[Choudhary 90] U. Choudhary and A. Sangiovanni-Vincentelli, "Constraint-based channel routing for analog and mixed analog/digital circuits," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 198-201, 1990.

[Chung 79] 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, pages 19-36, 1979.

[Cong 89] J. Cong, "Pin assignment with global routing," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 302-305, 1989.

[Cong 88] J. Cong and B. T. Preas, "A new algorithm for standard cell global routing" *IEEE Int. Conf. on Computer-Aided Design*, pages 176-180, November 1988.

[Cong 91] J. Cong, "A provable near-optimal algorithm for the channel pin assignment problem," *private communication.*

[Dai 85] W. -M. Dai, T. Asano and E. S. Kuh, "Routing region definition and ordering scheme for building-block layout," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-4, No. 3, pages 189-197, 1985.

[Dai 87a] W. -M. Dai, M. Sato, E. S. Kuh, "A dynamic and efficient representation of building-block layout," *Proc. 24th ACM/IEEE Design Automation Conf.*, pages 376-384, 1987.

[Dai 87b] W. -M. Dai, E. S. Kuh, "Global spacing of building-block layout," *Proc. VLSI '87*, pages 161-173, 1987.

[Dai 87c] W. -M. Dai and E. S. Kuh, "Simultaneous floor planning and global routing for hierarchical building-block layout," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6, No. 5, pages 828-837, 1987.

[Dai 87d] W. -M. Dai, H. H. Chen, R. Dutta, M. Jackson, E. S. Kuh, M. Marek-Sadowska, M. Sato, D. Wang and X. M. Xiong, "BEAR: a new building-block layout system," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 34-37, 1987.

[Dai 89] W. -M Dai, B. Eschermann, E.S. Kuh and M. Pedram, "Hierarchical placement and floorplanning in BEAR," *IEEE Trans. on Computer-Aided Design* Vol. 8, No. 12, pages 1335-1349, December 1989.

[Detjens 87] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang, "Technology mapping in MIS," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 116-119, 1987.

[Deutsch 76] D. N. Deutsch, "A 'dogleg' channel router," *Proc. 13th ACM/IEEE Design Automation Conf.*, pages 425-433, 1976.

[Donath 69] W. E. Donath, "Hierarchical structure of computers," *IBM T. J. Watson Research Center Report RC 2392*, March 1969.

[Donath 70] W. E. Donath, "Stochastic model of the computer logic design process," *IBM T. J. Watson Research Center Report RC 3136*, November 1970.

[Donath 79] W. E. Donath, "Placement and average interconnection lengths of computer logic," *IEEE Trans. on Circuits and Systems*, Vol. CAS-26, No. 4, pages 272-277, April 1979.

[Dong 89] S. K. Dong, J. Cong and C. L. Liu, "Constrained floorplan design for flexible blocks," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 488-491, November 1989.

[Dunlop 84] A.E. Dunlop, V.D. Agrawal, D.N. Deutsch, M.F. Jukl, P. Kozak and M. Wiesel, "Chip layout optimization using critical path weighting," *Proc. 21th ACM/IEEE Design Automation Conf.*, pages 133-136, 1984.

[El Gamal 81b] A. A. El Gamal, Z. A. Syed, "A stochastic model for interconnections in custom integrated circuits," *IEEE Trans. on Circuits and Systems* , Vol. CAS-28, pages 883-893, 1981.

[El Gamal 81a] A. A. El Gamal, "Two dimensional stochastic model for interconnections in master slice circuits," *IEEE Trans. on Circuits and Systems*, Vol. CAS-28, No. 2, pages 127-138, February 1981.

[El-Mansy 88] Y. A. El-Mansy and W. M. Siu, "MOS technology advances," *Handbook of Advanced Semiconductor Technology and Computer Systems*, G. Rabbat ed., Van Nostrand Reinhold Company, pages 229-259, 1988.

[Fiduccia 82] C.M. Fiduccia and R.M. Mattheyses, "A linear-time heuristic for improving network partitions," *Proc. 19th ACM/IEEE Design Automation Conf.*, pages 175-181, 1982.

[Fletcher 81] R. Fletcher, *Practical methods of optimization*, Vol. 2, John Wiley and Sons, Chichester, 1981.

[Fowler 85] C. Fowler, G.D. Hachtel and L. Roybal, "New algorithms for hierarchical place and route of custom VLSI," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 273-275, 1985.

[Garey 79] M. R. Garey and D. S. Johnson, *Computers and intractability*, W. H. Freeman and Company, 1979.

[Garrod 88] D. Garrod, R. A. Rutenbar and L. R. Carley, "Automatic layout of custom integrated circuits in ANAGRAM," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 544-547, 1988.

[Hachtel 88] G. Hachtel, M. Lightner, R. Jacoby, C. Morrison, P. Moceyunas, and D. Bostick, "BOLD: the boulder optimal logic design system," *Hawaii Int. Symp. on Systems Sciences*, 1988.

[Hakimi 64] S. L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," *Oper. Res.*, 12, pages 450-459, 1964.

[Hanson 88] D. Hanson, "Interconnection analysis," *Physical Design Automation of VLSI Circuits*, B. Preas and M. Lorenzetti editors, Benjamin/Cummings Publ., Menlo Park, CA, pages 47-48, 1989.

[Hashimoto 87] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proc. 8th Design Automation Workshop*, pages 155-163, 1971.

[Hauge 87] P. S. Hauge, R. Nair and E. J. Yoffa, "Circuit placement for predictable performance," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 88-91, 1987.

[Heinbuch 88] D.V. Heinbuch ed., *CMOS 3 Cell Library*, Addison-Wesley Publishing Company, 1988.

[Heller 77] W. R. Heller, W. F. Mikhail and W. E. Donath, "Prediction of wiring space requirements for LSI," *Proc. 14th ACM/IEEE Design Automation Conf.*, pages 32-42, June 1977.

[Herrigel 89] A. Herrigel, M. Glaser and W. Fichtner, "A global floorplanning technique for VLSI layout," *IEEE Int. Conf. on Computer Design*, pages 92-95, 1989.

[Hitchcock 82] R. B. Hitchcock, G. L. Smith and D. D. Cheng, "Timing analysis of computer hardware," *IBM J. Res. Develop.*, Vol. 26, No. 1, pages 100-105, January 1982.

[Hsh 87] Y. C. Hsh and W. J. Kubitz, "A procedure for chip floor planning," *Proc. Int. Symp. on Circuits and Systems*, pages 568-571, 1987.

[Jackson 87] M. A. B. Jackson, E. S. Kuh and M. Marek-Sadowska, "Timing-driven routing for building block layout," *Proc. Int. Symp. on Circuits and Systems*, pages 518-519, 1987.

[Jackson 89] M. A. B. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's," *Proc. 26th ACM/IEEE Design Automation Conf.*, pages 370-375, 1989.

[Jackson 90] M. A. B. Jackson, A. Srinivasan and E. S. Kuh, "A fast algorithm for performance-driven placement," *IEEE Inl. Conf. on Computer-Aided Design*, pages 328-331, 1990.

[Keutzer 87] K. Keutzer, "DAGON: technology binding and local optimization by DAG matching," *Proc. 24th ACM/IEEE Design Automation Conf.*, pages 341-347, 1987.

[Khellaf 87] M. Khellaf, "On the partitioning of graphs and hypergraphs," Ph.D. dissertation, Dept. IEOR, Univ. Calif., Berkeley, 1987.

[Kleinhans 90] J. M. Kleinhans, G. Sigl, F. M. Johannes and K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. on Computer-Aided Design*, Vol. 10, No. 3, pages 356-365, March 1991.

[Koren 72] N.L. Koren, "Pin assignment in automated printed circuit board," *Proc. 9th Design Automation Workshop*, pages 72-79, 1972.

[Kurdahi 86] F.J. Kurdahi and A.C. Parker, "PLEST: a program for area estimation of VLSI integrated circuits," *Proc. 23rd ACM/IEEE Design Automation Conf.*, pages 467-473, 1986.

[Kurdahi 89] F. J. Kurdahi and A. C. Parker, "Techniques for area estimation of VLSI layouts," *IEEE Trans. on Computer-Aided Design*, Vol. 8, No. 1, pages 81-92, January 1989.

[Landman 71] B. S. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Trans. on Computers*, C-20, No. 12, pages 1469-1479, December 1971.

[La Potin 86] D. P. La Potin and S. W. Director, "Mason: a global floorplanning approach for VLSI design," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-5, No. 4, pages 477-489, October 1986.

[Lauther 79] U. Lauther, "A min-cut placement algorithm for general cell assemblies based on a graph representation," *Proc. 16th ACM/IEEE Design Automation Conf.*, pages 1-10, 1979.

[Lauther 85] U. P. Lauther, "Channel routing in a general cell environment," *Proc. VLSI-85*, 1985.

[Lee 88] K. W. Lee and C. Sechen, "A new global router for row-based layout," *IEEE Inl. Conf. on Computer-Aided Design*, pages 180-183, November 1988.

[Lengauer 90] T. Lengauer, *Combinatorial algorithms for integrated circuit layout*, Wiley-Teubner Series in Computer Science, 1990.

[M-Sadowska 86] M. Marek-Sadowska, "Route planner for custom chip design," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 246-249, 1986.

[M-Sadowska 89] M. Marek-Sadowska and S. P. Lin, "Timing driven placement," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 94-97, 1989.

[Mayrhofer 90] S. Mayrhofer and U. Lauther, "Congestion driven placement using a new multi-partitioning heuristic," *IEEE Int. Conf. on Computer-Aided Design*, pages 332-335, 1990.

[Mayrhofer 91] S. Mayrhofer, M. Pedram and U. Lauther, "A flow-based approach to the placement of Boolean networks," *Proc. VLSI-91*, 1991.

[MCNC 88] "Logic synthesis and optimization benchmarks - user guide," *Microelectronics Research Center of North Carolina*, 1988.

[MCNC 90] "Floorplanning benchmarks," *MCNC Int. Workshop on Layout Synthesis,* Research Triangle Park, NC, 1990.

[Murgai 90] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli, "Logic synthesis for programmable gate arrays," *Proc. 27th ACM/IEEE Design Automation Conf.*, pages 620-625, 1990.

[Murofushi 90] M. Murofushi, M. Yamada and T. Mitsuhashi, "FOLM-planner: a new floorplanner with a frame overlapping floorplan model," *Proc. IEEE Int. Conf. on Computer-Aided Design*, 1990.

[Nair 89] R. Nair, C. L. Berman, P. S. Hauge and E.J. Yoffa, "Generation of performance constraints for layout," *IEEE Trans. on Computer-Aided Design*, Vol. 8,No. 8, pages 860-874, August 1989.

[Ogawa 86] Y. Ogawa, T. Ishii, Y. Shiraishi, H. Terai, T. Kozawa, et al., "Efficient placement algorithms optimizing delay for high-speed ECL masterslice LSI's," *Proc. 23th ACM/IEEE Design Automation Conf.*, pages 404-410, 1986.

[Ogawa 90] Y. Ogawa, M. Pedram and E.S. Kuh, "Timing-driven placement for general cell layouts," *Proc. Int. Symp. on Circuits And Systems*, Vol. 2, pages 872-875, May 1990.

[Otten 82] R. H. J. M. Otten, "Automatic floorplan design," *Proc. 19th ACM/IEEE Design Automation Conf.*, pages 261-267, 1982.

[Otten 83] R. H. J. M. Otten, "Efficient floorplan optimization," *Proc. IEEE Int. Conf. on Computer Design*, pages 499-502, 1983.

[Ousterhout 84] J.K. Ousterhout, "Corner stitching: a data structuring technique for VLSI layout tools," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-3, 1984.

[Pedram 89a] M. Pedram and B. T. Preas, "Accurate prediction of physical design characteristics of random logic," *IEEE Int. Conf. on Computer Design*, pages 100-108, 1989.

[Pedram 89b] M. Pedram and B. T. Preas, "Interconnection length estimation for optimized standard cell layouts," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 390-393, 1989.

[Pedram 90a] M. Pedram and B.T. Preas, "A hierarchical floorplanning approach," *Proc. Int. Conf. on Computer Design*, pages 332-338, 1990.

[Pedram 90b] M. Pedram, M. Marek-Sadowska and E. S. Kuh, "Floorplanning with pin assignment," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 98-101, 1990.

[Pedram 90c] M. Pedram, W. M. Dai, M. Marek-Sadowska, G. Carvalho, D. Wang and B. Chen, "BEAR-FP Manual, distribution 1.0," *Memorandum N. UCB/ERL M90/118*, December 1990.

[Pedram 91a] M. Pedram and N. Bhat, "Layout driven technology mapping," *Proc. 28th ACM/IEEE Design Automation Conf.*, pages 99-105, 1991.

[Pedram 91b] M. Pedram, K. Chaudhary and E. S. Kuh, "I/O pad assignment based on circuit structure," *To appear in Proc. IEEE Int. Conf. Computer Design*, 1991.

[Pedram 91c] M. Pedram and N. Bhat, "Layout driven logic restructuring / decomposition," *To appear in Proc. IEEE Int. Conf. Computer-Aided Design*, 1991.

[Prasitju 89] S. Prasitjutrakul and W. J. Kubitz, "Path-delay constrained floorplanning: a mathematical programming approach for initial placement," *Proc. 26th ACM/IEEE Design Automation Conf.*, pages 364-369, 1989.

[Preas 87] B. T. Preas, "Benchmarks for cell-based layout systems," *Proc. 24th ACM/IEEE Design Automation Conf.*, pages 319-320, 1987.

[Reed 85] J. Reed, "YACR: yet another channel router," Master's Report, University of California, Berkeley, February 1985.

[Rudell 89] R. Rudell, "Logic synthesis for VLSI design," Ph.D. dissertation, University of California, Berkeley, 1989.

[Sakurai 83] T. Sakurai, "Approximation of wiring delays in MOSFET LSI," *IEEE Journal of Solid-State Circuits*, Vol. SC-18, No. 4, pages 418-426, August 1983.

[Saraswat 82] 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, pages 645-650, 1982.

[Sastry 86] S. Sastry and A. C. Parker, "Stochastic models for wirability analysis of gate arrays," *IEEE Trans. on Computer-Aided-Design*, Vol. CAD-5, No. 1, pages 52-65, January 1986.

[Savoj 91] H. Savoj, H. -Y. Wang, "Improved scripts in MIS-II for logic minimization of combinational circuits," *Proc. Int. Workshop on Logic Synthesis,* Vol. 3, 1991.

[Schittkowski] K. Schittkowski, "Computational mathematical programming," *NATO ASI Series,* Vol. 15, Springer Verlag.

[Sechen 85] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE J. of Solid State Circuits*, Vol. 20, No. 2, pages 510-522, April 1985.

[Sechen 87a] C. Sechen and K. W. Lee, "An improved simulated annealing algorithm for row-based placement," *Proc. IEEE Inl. Conf. on Computer-Aided Design*, pages 478-481, November 1987.

[Sechen 87b] C. Sechen, "Average interconnection length estimation for random and optimized placements," *IEEE Inl. Conf. on Computer-Aided Design*, pages 190-193, November 1988.

[Sechen 88] C. Sechen, "VLSI placement and global routing using simulated annealing," *Kluwer Academic Publishers*, 1988.

[Srinivasan 91] A. Srinivasan, K. Chaudhary and E. S. Kuh, "RITUAL: an algorithm for performance-driven placement of cell-based ICs," *Proc. Third Physical Design Workshop*, May 1991.

[Stockmeyer 83] L. Stockmeyer, "Optimal orientation of cells in slicing floorplan designs," *Information and Control*, Vol. 57, pages 91-101, 1983.

[Szepieniec 86] A. A. Szepieniec, "Integrated placement / routing in sliced layouts," *Proc. 23rd ACM/IEEE Design Automation Conf.*, pages 300-307, 1986.

[Teig 86] S. Teig, R.L. Smith and J. Seaton, "Timing-driven layout of cell-based ICs," *VLSI Systems Design*, pages 63-73, May 1986.

[Touati 90] H. J. Touati, C. W. Moon, R. K. Brayton and A. Wang, "Performance-oriented technology mapping," *Proc. 6th MIT Conf, Advanced Research in VLSI*, W. J. Dally ed., pages 79-97, 1990.

[Touati 91] H. J. Touati, H. Savoj, and R. K. Brayton, "Delay optimization of combinational logic circuits by clustering and partial collapsing," *To appear in Proc. IEEE Int. Conf. Computer Design*, 1991.

[Tsay 88] R. S. Tsay, E. S. Kuh and C. P. Hsu, "Proud: A sea-of-gates placement algorithm," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 318-323, 1988.

[Tuck 90] B. Tuck, "High-density gate arrays: products too far ahead of technology?," *Computer Design*, pages 71-72, August 1990.

[Ueda 85] K. Ueda, H. Kitazawa and I. Harada, "CHAMP: chip floor plan for hierarchical VLSI layout design," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-4, pages 12-22, 1985.

[van Ginneken 90] L. P. P van Ginneken and R. J. M. Otten., "Optimal slicing of plane point placements," *Proc. 1st European Design Automation Conf.*, pages 322-326, 1990.

[Vijayan 90] G. Vijayan and R. S. Tsay, "Floorplanning by Topological Constraint Reduction," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 106-110, 1990.

[Wang 90] D. Wang, "Ring routing for macro-cell layout," *Proc. 27th ACM/IEEE Design Automation Conf.*, pages 193-198, 1990.

[Wallace 90] D. E. Wallace, M. S. Chandrasekhar, "High-level delay estimation for technology independent logic equations," *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 188-191, 1990.

[Wimer 88] S. Wimer, I. Koren and I. Cederbaum, "Optimal aspect ratios of building blocks in VLSI," *Proc. 25th ACM/IEEE Design Automation Conf.*, pages 66-72, 1988.

[Wipfler 85] G. J. Wipfler, D. A. Mlynski and H. Hillner, "An automatic placement procedure for integrated circuits," *Proc. Int. Symp. on Circuits and Systems*, pages 13-16, 1985.

[Wong 89a] D. F. Wong and P. Sakhamuri, "Efficient floorplan area optimization," *Proc. 26th ACM/IEEE Design Automation Conf.*, pages 586-589, 1989.

[Wong 89b] D. F. Wong and K. The, "An algorithm for hierarchical floorplan design," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 484-487, 1989.

[Xiong 87] X. M. Xiong and E. S. Kuh, "Nutcracker: an efficient and intelligent channel spacer," *Proc. 24th ACM/IEEE Design Automation Conf.*, pages 298-304, 1987.

[Xiong 88] X. M. Xiong and E. S. Kuh, "The constraint via minimization problem for PCB and VLSI design," *Proc. 25th ACM/IEEE Design Automation Conf.*, pages 573-578, 1988.

[Yao 88] X. Yao, M. Yamada and C.L. Liu, "A new approach to the pin assignment problem," *Proc. 25th ACM/IEEE Design Automation Conf.*, pages 566-572, 1988.

[Yao 90] X. Yao and C.L. Liu, "Pin position assignment for movable pins in macro-cells," *Int. Journal Computer-Aided VLSI Design*, 1990.

[Yonezawa 90] N. Yonezawa, N. Nishiguchi, A. Etani, F. Tsukuda, R. Hashishita, "A VLSI floorplanner based on 'balloon' expansion," *Proc. 1st European Design Automation Conf.*, pages 257-262, 1990.

[Zimmerman 86] G. Zimmerman, "Top-down design of digital systems," E. Horbst, editor, *Advances in CAD for VLSI,* Vol. 2, North-Holland, New York, pages 185-206, 1986.

[Zimmerman 88] G. Zimmerman, "A new area and shape function estimation technique for VLSI layouts," *Proc. 25th ACM/IEEE Design Automation Conf.*, pages 60-65, 1988.