6-1994

Floorplanning for Mixed Block and Cell Designs

Arun G. Shanbhag

FLOORPLANNING FOR MIXED BLOCK AND CELL DESIGNS

by

Arun G. Shanbhag

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
June 1994

To

My Uncle, Manjunath Nayak
and
My Parents

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my advisor, Professor Naveed A. Sherwani, from whom I have learned many valuable skills, both research and otherwise. His consistent support, accessibility and his constant motivation, has been a source of encouragement for me. I expect our professional working relationship to continue for many years to come.

Professor Donald Nelson, the Department Chair, has my gratitude for his continued support in providing research facilities that has been a great help in finishing this thesis work and conducting other research.

I am grateful to the members of my thesis committee, Professor Alfred Boals and Professor Ajay Gupta, who accepted this additional task with high enthusiasm.

I would like to thank both the old and the new 'nitegroup' members. Jahangir Hashmi, Moazzem Hossain, Siddharth Bhingarde and Surendra Burman, for their initial help, cooperation, and friendship. My sincere gratitude to Srinivasa, Praveen and Tim for their help on "ARCHITECT". I would also like to thank Sandeep, Rajen, Shreekrishna, John and Aman for all their help and cheerful company during the many long nights that we spent working together in the VLSI lab.

I would also like to thank all my friends at WMU, who, in one way or the other, made my study here an enjoyable one.

Special thanks to Suzanne Moorian and Phyllis Wolf from Computer Science Department, who helped me in many administrative situations: their pleasant characters and helpful personalities are assets to us all.

Finally, I thank my parents for their encouragement, guidance and support which has always helped me in accomplishing my goals.

Arun G. Shanbhag

FLOORPLANNING FOR MIXED BLOCK AND CELL DESIGNS

Arun G. Shanbhag, M.S.

Western Michigan University, 1994

Floorplanning is one of the important phases of the VLSI Physical Design cycle. The quality of a floorplan is usually not evident until the routing phase. A bad floorplan can lead to an unroutable design requiring another iteration of the floorplanning phase. Use of over-the-cell routing has led to zero routing footprints. Any further reduction in area of the layout is possible only by reducing the white space or the vacant space from the floorplan, that is by improving the floorplan of the layout.

Currently, the Mixed Block and Cell (MBC) design style is gaining popularity. This evolving design style is generally used to design high performance microprocessor chips. In the last three to four years, most of the microprocessors have been designed using this design style.

In this thesis, we develop a floorplanning scheme for Mixed Block and Cell designs which differs radically from all existing approaches. Existing algorithms, deal with rectangular shaped flexible blocks while generating the floorplans. In our algorithm, we exploit the flexibility of the standard cell regions by assigning rectilinear shapes to these regions. This improves not only the area of the layout but can also improve the performance of the design. We have developed an interactive tool, called "ARCHITECT" based on our algorithm, for floorplanning of MBC designs.

TABLE OF CONTENTS

Table of Contents - continued

LIST OF TABLES

LIST OF FIGURES

List of Figures - continued

CHAPTER I

INTRODUCTION

Floorplanning and Placement are important steps of the VLSI Physical Design cycle. Both these steps are concerned with selecting good layout alternatives for each block, as well as the enitre chip. During placement, the exact locations of the blocks on the chip is determined. The objective of a placement algorithm is to find a minimum area arrangement for the blocks such that all the interconnections required between the blocks can be carried out. Usually, placement is done in two phases. In the first phase an initial placement of the blocks is generated. The second phase is usually iterative and improves the initial placement until the layout has minimum area and conforms to design specifications. The input to the placement phase is a set of blocks, the number of terminals for each block and the netlist. If the layout of the circuit within a block has been completed then the dimensions of the block are also known. The blocks for which the dimensions are known are called *fixed* blocks and the blocks for which dimensions are yet to be determined are called *flexible* blocks. Thus during the placement phase, we need to determine an appropriate shape for each block (if shape is unknown), location of each block on the layout surface, and determine the locations of pins on the boundary of the blocks. The problem of assigning locations to the fixed blocks on a layout surface is called the *Placement* problem. If some or all of the blocks are flexible then the problem is called the *Floorplanning* problem. Hence, the placement problem is a restricted version of the floorplanning problem. The terminology is slightly confusing as floorplanning problems are placement problems as well but these terminologies have been widely used and accepted. Floorplanning sets up the ground work for a good layout. However, it is computationally quite hard. Very often the task of floorplan layout is done by a design engineer,

rather than by a CAD tool. Both the floorplanning and routing stages are quite critical as the quality of the layout is not evident until the routing phase. A bad floorplan may lead to an unroutable design thereby requiring another iteration of the floorplanning phase. The performance of the chip depends heavily on the floorplan generated.

Usually, several factors are considered by the placement and floorplanning algorithms. These factors are discussed below:

1. Block Shapes: The blocks are assumed to be rectangular in shape. This assumption is made in order to simplify the problem. The shapes resulting from the floorplanning algorithms are mostly rectangular for the same reason. Floorplanning algorithms use *aspect ratios* for determining the shape of a block. The aspect ratio of a block is the ratio of the height to the width of the block. Usually there is an upper and a lower bound on the aspect ratios, restricting the dimensions that the block can have. More recently, other shapes such as L-shapes have been considered, however dealing with such shapes is computationally hard.

2. Routing consideration: In order to generate a layout which is routable, the placement and floorplanning algorithms need to estimate area required for routing. The blocks are placed in a manner such that there is sufficient routing area between the blocks so that routing algorithms can successfully complete the routing within these areas.

3. High Performance circuits: For high performance circuits the blocks are to be placed such that all critical path lengths are minimized.

Placement and Floorplanning

The placement problem can be stated as follows: Given an electrical circuit consisting of fixed blocks, and a netlist interconnecting terminals on the periphery of these blocks and on the periphery of the circuit itself, construct a layout indicating the positions of each block such that all the nets can be routed and

the total layout area is minimized. The objective for high performance systems is to minimize the total delay of the system, by minimizing the lengths of the critical paths. The quality of a placement can be judged by the area of the layout, completion of routing and circuit performance.

The placement problem can be formally stated as follows: Let $B_1, B_2, \ldots, B_n$, be the blocks to be placed on the chip. Each $B_i, 1 \leq i \leq n$, has associated with it a height h_i, and a width w_i. Hence, each block B_i, can be represented by a rectangle R_i, with the coordinates of its lower left corner specified by $(x_i, y_i), 1 \leq i \leq n$. Let O_i, represent the orientation of block B_i. $O_i = 1$, if the block is rotated by 90^o, else $O_i = 0$. Let $\mathcal{N} = \{N_1, N_2, N_3, \ldots, N_m\}$, be the set of nets representing the interconnection between different blocks. Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_k\}$, represent rectangular empty areas allocated for routing between blocks. Let L_i, denote the estimated length of net N_i, $1 \leq i \leq m$. The placement problem is to find the location (x_i, y_i) and orientation O_i for each rectangle R_i, representing block $B_i, 1 \leq i \leq n$, such that

1. No two rectangles overlap, that is $R_i \cap R_j = \phi$, $1 \leq i, j \leq n$,

2. Placement is routable, that is $Q_j, 1 \leq j \leq k$, is sufficient to route all the nets.

3. The total area of the rectangle bounding $\mathcal{R}$ and $\mathcal{Q}$ is minimized.

4. The total wirelength is minimized, that is $\sum_{i=1}^{m} L_i$ is minimized. In the case of high performance circuits, the length of longest net $\max\{L_i \mid i = 1, \ldots, m\}$ is minimized.

Floorplanning is the placement of flexible blocks, that is blocks with fixed area but unknown dimensions. It is a much more difficult problem as compared to the placement problem. In floorplanning, several layout alternatives for each block are considered. Usually, the blocks are assumed to be rectangular and the lengths and widths of these blocks are determined in addition to their locations. The blocks are assigned dimensions by making use of the aspect ratios. Usually,

there is an upper and a lower bound on the aspect ratio a block can have as the blocks cannot take shapes which are too long and very thin. Initial estimate on the set of feasible alternatives for a block can be made by statistical means, *i.e.*, by estimating the expected area requirement of the block. Many techniques of general block placement have been adapted to floorplanning.

As the placement problem is a restricted version of floorplanning all the constraints and objective functions applicable for the placement problem are applicable. Usually, the input consists of $B_1, B_2, \ldots, B_n$ circuit blocks, with area $a_1, a_2, \ldots, a_n$, respectively. Associated with each block are two aspect ratios A_i^l and A_i^h, which give the lower and the upper bound on the aspect ratio for the block. The floorplanning algorithm has to determine the width w_i and height, h_i of each block B_i such that $A_i^l \leq \frac{h_i}{w_i} \leq A_i^h$. In addition to finding the shapes of the blocks, the floorplanning algorithm has to generate a valid placement such that the area of the layout is minimized.

Design Styles

Placement and floorplanning algorithms are generally developed for a particular design style. The two most popular design styles are the full-custom design style and the standard cell design style. The full-custom design style consists of blocks of different sizes. In full-custom designs, blocks can be placed at any location on the chip surface without any restrictions which allows very compact layouts. However, the process of automating a full-custom design style has a much higher complexity than other restricted models. Hence, this design style is used only when minimum area layouts are required and there is no constraint on the designing time required. The space not occupied by the blocks is used for routing interconnecting wires.

On the other hand, the design process in a standard cell design style is somewhat simpler than a full-custom design style. Standard cell architecture considers the layout to consist of rectangular cells of same height. Several predefined cells are available in a standard cell library. The cells are placed in rows and the space between two rows, called a *channel* is used to perform interconnections between the cells. The interconnections for cells which lie in two non-adjacent rows and need to be connected is done by passing the wire in an empty space between two cells in a row. This empty space is called a *feedthrough.* This design style is well-suited for moderate size circuits and medium production volumes. Physical design using standard cells is somewhat simpler compared to full-custom and efficient using modern design tools.

Recently, a new kind of design style called the Mixed Block and Cell (MBC) design style has been gaining popularity. This evolving design style consists of a mix of both blocks and standard cells. Usually in an MBC design, regular structures such as RAM and datapath blocks are implemented as full-custom macro cells and other logic is implemented using standard cells. The MBC design style offers a compromise between the performance of a full custom design style to the relative simplicity of layout generation problem of the standard cell design style. The time to market required for MBC designs is also considerably less compared to that of the full custom design style as it allows reuse of macro blocks. Hence, the MBC design style is being preferred over other design styles. In an MBC design, usually, the macro blocks are taken from existing designs and the additional functionality is developed using standard cells. The macro blocks are also called as *fixed* blocks as their areas, shapes and pin locations are well defined. On the other hand, the functional blocks build using the standard cells are highly flexible enough to be molded into a variety of different rectilinear shapes. Hence these blocks are referred to as *juice* or *flexible* blocks. Typically, on a chip designed

with the MBC design style, there are about 10 fixed blocks and about 10,000 standard cells. Figure 1 shows the layout of a MBC design.

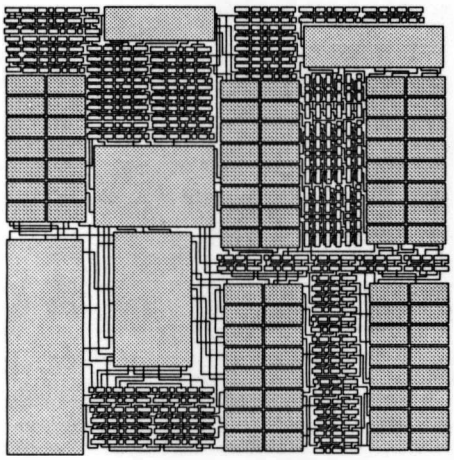

Figure 1. A Mixed Block and Cell Layout.

Floorplanning for MBC Designs

In this thesis, we consider the floorplanning problem for MBC designs. We have developed a heuristic algorithm called Flexible Block Reshaping (FBR) algorithm, which has been incorporated into "ARCHITECT", a floorplanning tool for MBC designs. The floorplanning problem for MBC designs can be stated formally as follows: Let $B_{f_1}, B_{f_2}, B_{f_3}, \ldots, B_{f_n}$, be the set of fixed blocks. Each fixed block B_{f_i}, is represented as a rectangle R_{f_i}, $1 \leq i \leq n$, with width w_{f_i} and height h_{f_i}, $1 \leq i \leq n$. Let O_{f_i}, denote the orientation of the fixed block B_{f_i} and (x_{f_i}, y_{f_i}) denote the coordinates of the lower left corner of the block. Let $B_{v_1}, B_{v_2}, B_{v_3}, \ldots, B_{v_m}$, be the set of flexible blocks which are represented as a set of rectangles R_{v_i}, $1 \leq i \leq m$. Each flexible block B_{v_i}, actually consists of a set of standard cells $\mathcal{S}_{v_i}$, $1 \leq i \leq m$. Each flexible block B_{v_i}, has an area A_{v_i} and the orientation of the block is denoted by O_{v_i}, $1 \leq i \leq m$. Associated with

each flexible block B_{v_i}, are two aspect ratios, $a_{v_i}^{min}$ and $a_{v_i}^{max}$ which represent the lower and the upper bound respectively on the aspect ratio of the block B_{v_i}. The coordinates of the lower left corner of a flexible block B_{v_i} are given by (x_{v_i}, y_{v_i}). Let $\mathcal{N} = \{N_1, N_2, N_3, \ldots, N_m\}$, be the set of nets representing the interconnection between the different blocks. Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_k\}$, represent rectangular vacant areas or white space in the floorplan. Let L_i, denote the estimated length of net N_i, $1 \leq i \leq m$. The floorplanning problem for MBC designs is

1. To determine the location (x_{f_i}, y_{f_i}) and the orientation O_{f_i}, for each rectangle R_{f_i}, representing a fixed block $B_i, 1 \leq i \leq n$

2. To determine the location (x_{v_i}, y_{v_i}), the orientation O_{v_i} and an aspect ratio a_{v_i}, for each rectangle R_{v_i}, representing a flexible block $B_{v_i}, 1 \leq i \leq m$

such that:

1. $a_{v_i}^{min} \leq a_{v_i} \leq a_{v_i}^{max}$ for each flexible block $B_{v_i}, 1 \leq i \leq m$,
2. No two rectangles overlap,
3. The vacant area $\mathcal{Q}$, is minimized,
4. The total wirelength is minimized, that is $\sum_{i=1}^{m} L_i$ is minimized.

After the floorplan is generated, the placement of standard cells $\mathcal{S}_{v_i}$, within the region specified by $R_{v_i}, 1 \leq i \leq m$, has to be determined. This is the traditional problem formulation for the MBC floorplanning problem. Several algorithms [1, 8, 9, 10, 11, 12, 13] have been developed for the traditional problem. In the traditional problem, the shapes of flexible blocks are restricted to rectangular in order to simplify the problem. The flexibility of the standard cell region is not exploited because of the restriction on its shape. The standard cell regions can be used to eliminate the vacant or white spaces in a floorplan. This can be done only if rectilinear shapes are considered for flexible blocks. This is the main difference between our approach and the traditional approach. In our approach, we generate rectilinear shapes for flexible blocks. In order to control the shapes of these blocks, we have defined two new shape control parameters.

We have incorporated our floorplanning algorithm in "ARCHITECT", a tool for floorplanning of MBC designs. Our results indicate that the generation of rectilinear shapes for flexible blocks leads to an improvement in the floorplan areas for MBC designs.

Chapter II of this thesis has been devoted to the discussion of the various existing techniques used for floorplanning. We have presented a classification scheme for existing techniques. In Chapter II, we also discuss the existing approaches for floorplanning of MBC designs. In Chapter III, we give the motivation behind developing a new floorplanning algorithm and discuss more about the shape parameters used for controlling the rectilinear shapes generated by our algorithm. Chapter IV, gives an overview of the ARCHITECT floorplanning tool. The details of our floorplanning algorithm are described in Chapter V. Experimental results and conclusions are presented in Chapter VI.

CHAPTER II

AN OVERVIEW OF EXISTING FLOORPLANNING TECHNIQUES

In this chapter we give an overview of existing techniques for floorplanning. We will provide a scheme for classification of the various existing algorithms and briefly discuss one algorithm in each of these broad areas. As the focus of this thesis is on MBC designs, we will discuss existing floorplanning algorithms for MBC designs.

Classification of Floorplanning Algorithms

In this section, we present a classification scheme for existing floorplanning algorithms. The floorplanning algorithms can be classified into three broad areas depending on the approach used. These three areas are:

1. Constraint based methods.
2. Rectangular dualization based methods.
3. Hierarchical tree based methods.

We will discuss each of these categories in more details in the following sections.

Constraint Based Methods

The constraint based methods or algorithms make use of certain constraints that are either derived from the given problem or that are defined as input to the problem. The constraint based algorithms are iterative improvement type of floorplanning algorithms. At any given time, the solution generated is guaranteed to satisfy the constraints specified by the constraint set. There are several very well known algorithms in this area. The algorithm proposed by Vijayan and Tsay [1]

uses topological constraints while generating floorplans. Another approach developed by Sutanthavibul, Shragowitz and Rosen [2] uses an integer programming formulation for generating the floorplan. The floorplanning problem is modeled as a set of linear equations using 0/1 integer variables. Constraints considered include overlap constraints and routability constraints. The overlap constraints prevent any two blocks from overlapping whereas the routability constraints estimate the routing area required between the blocks. S. Dong, J. Cong and C. Liu [3] have presented a floorplan design algorithm which is capable of sizing a set of flexible blocks and placing them according to given constraints on relative positions as well as separation requirements in both horizontal and vertical directions.

The method, proposed by Vijayan and Tsay [1], constructs a floorplan of optimal area that satisfies (respects) a given set of constraints. A set of horizontal and vertical topological (*i.e.*, ordering) constraints is derived from the relative placement of blocks. Given a constraint set, it is usually the case that there is no reason to satisfy all the constraints in the set. This is especially true when a majority of the blocks have flexible shapes. A floorplan is said to respect a constraint, if for each pair of blocks, the floorplan satisfies at least one constraint (horizontal or vertical). A constraint set is said to be *overconstrained* if it has many redundant constraints. It is desirable to derive a complete constraint set from the input relative placement and then to remove those redundant constraints that result in reduction of floorplan area.

A topological constraint set of a set of blocks is given by two directed acyclic graphs (G_H, G_V): G_H is the horizontal constraint graph and G_V is the vertical constraint graph. In order to reduce the floorplan area, the heuristic iteratively removes a redundant constraint from the critical path of either G_H or G_V and also iteratively reshapes the blocks on the critical paths of the two graphs. Critical path is the longest path in G_H or G_V. The *input* to the algorithm is a constraint

set (G_H, G_V) of the set of blocks. To minimize the floorplan area, repeat steps 1 and 2 until there is no improvement in the floorplan area.

Step 1: Repeat the following steps until are no strongly redundant edges on P_H or P_V exist. G_H and G_V are topologically sorted and swept. Either P_H or P_V, whichever is more critical is selected, where P_H and P_V are the critical paths of G_H and G_V respectively. The strongly redundant edge on the selected critical path is eliminated.

Step 2: The current shapes of the blocks are stored and a path, either P_V or P_H is selected depending on which of the two is more critical. All the flexible blocks on the selected path are reshaped. G_H and G_V are scanned again to construct the new floorplan. If the newly generated floorplan is better than the previous one the stored block shapes are updated. All the steps described above are repeated, a specified number of times.

Each pass of the algorithm constitutes one execution of two steps. Constraint reduction takes place in step 1 and step 2 does the reshaping of the blocks. If the chip dimensions are fixed, the passes are repeated until the target dimensions are reached. Otherwise the passes can be repeated until there is improvement in the floorplan area. Typically three or four passes are required.

The purpose of removing a redundant edge on the critical path is to break the path into two smaller paths. A good choice for such a redundant edge is the one which is nearest to the center point of the path. The above heuristic removes only one redundant constraint from a critical path at each iteration, and thus seeks to minimize the number of constraints removed. An edge can be checked for strong redundancy in constant time if we maintain the adjacency matrix of G_H and G_V. It takes $O(n^2)$ time to set up adjacency matrices. A topological sort of a directed acyclic graph with n nodes and m edges takes $O(m+n)$ time. The number of topological sorts executed depends on the number of redundant edges

removed, the user-specified value for the number of reshaping iterations, and the number of passes.

Rectangular Dualization Based Methods

Rectangular dualization method first originated in the field of computer aided architectural design. This method has been the subject of extensive research over the last decade for its application to chip floorplanning. Given a floorplan F, a *rectangular graph*, $G(V, E)$, represents the adjacency of blocks in F. Each vertex in V corresponds to a distinct block in F and there is an undirected edge (u, v) in E iff the block corresponding to vertex u and that corresponding to vertex v are adjacent. It is easy to see that for a given floorplan a unique rectangular graph always exists. By definition, a rectangular floorplan is embedded on a plane which implies that its rectangular graph is a *plane graph.* The assumption about all intersections except the corners of F being T-junctions leads to the fact that all the internal faces of G are bounded by three edges. Thus, a rectangular graph is a *plane triangulated graph.* Plane triangulated graphs depicting adjacency of blocks are also known as neighborhood graphs. Given an n-vertex plane triangulated graph G, its rectangular dual RD consists of n non-overlapping rectangles where each vertex i in G corresponds to a distinct rectangle in RD. An edge (i, j) in G, requires rectangles i and j to be adjacent in RD. The rectangular dual of G, if it exists corresponds to a valid rectangular floorplan where the rectangles represent the the blocks in the floorplan.

Kozminski and Kinnen [4] established necessary and sufficient conditions for the existence of a rectangular dual when a graph is planar and satisfied the following properties:

P1: Every face (except the exterior) is a triangle.

P2: All internal vertices have degree ≥ 4.

P3: All cycles that are not faces have length ≥ 4.

Using their necessary and sufficient conditions, Kozminski and Kinnen developed an algorithm to obtain a rectangular dual, when one existed, in $O(n^2)$ time, where n is the number of vertices in the graph. Bhaskar and Sahni [5], proved that only conditions **P1** and **P3** are necessary and sufficient and they developed an $O(n)$ algorithms to obtain a rectangular dual.

In order to modify a given logical network to have a rectangular dual, the following steps are necessary. First, the network has to be planarized by deleting a minimum number of edges or a set of edges with minimum total weight, which has been proved to be NP-Complete. Second, the resulting planar graph has to be triangulated to generate a plane triangulated graph, which can be done in linear time. Third, all complex triangles have to be eliminated from the plane triangulated graph, which is an open problem. Finally, the proper plane triangulated graph has to be dualized yielding a dissection of a rectangle into rectangles - a floorplan of the network. For a given proper plane triangulated graph, there are many rectangular duals. If the rectangular dualization method is applied to a large network, the complexity is too high to be practical; if it is applied to a small network, we might as well enumerate all possible floorplans. Hence this technique is not very practical.

Hierarchical Tree Based Methods

Hierarchical tree based methods represent a floorplan as a tree. Each leaf in the tree corresponds to a block and each internal node corresponds to a composite block in the floorplan. A floorplan is said to be *hierarchical* of *order* k, if it can be obtained by recursively partitioning a rectangle into at most k parts. Physical hierarchy can be generated in two ways: top-down partitioning or bottom-up clustering. Partitioning assumes that the relative areas (or number of nodes) Within partitions, at a given level of hierarchy, may be fixed during a top down construction of a decomposition tree (or partitioning tree). There

is no justification, except convenience, for this assumption. The optimal choice of relative areas varies from problem instance to problem instance, but there is no way to determine a desirable ratio, in top-down construction. Placements performed by min-cut method, a popular partitioning algorithm, often creates lot of vacant space or white space. Clustering on the other hand is a bottom-up algorithm for constructing a decomposition tree (or cluster tree).

In [6] a hierarchical floorplanner for arbitrary size rectangular blocks using the clustering approach has been proposed. At each level of the hierarchy, highly connected blocks (or clusters of blocks) are grouped together into larger clusters. At each level, the number of blocks is limited to five so that simple pattern enumeration and exhaustive search algorithms can be used later. Blocks (or block clusters) which are connected by edges of greater than average edge weight are grouped into a single cluster, if the resulting cluster has less than five members.

After forming the hierarchical clustering tree, a floorplanner and a global router together perform a top-down traversal of the hierarchy. Given an overall aspect ratio goal and I/O pin goal, at each level of the hierarchy, the floorplanner searches a simple library of floorplan templates and considers all possible room assignments which meet the combined goals of aspect ratios and I/O pins. At each level, the global routing problem is formulated as a series of minimum steiner tree problems in partial 3-trees. The global routing solution at the current level is used as the I/O pin goal for the floorplan evaluation, and as base for the global routing refinement at the next level. This floorplanning and global routing create constraints on the aspect ratio of the rooms, and gives assignments of I/O pins on the walls of the rooms, which are recursively transmitted downward as subgoals to the floorplanner and global router. While evaluating the cost of a given floorplan template and room assignment, both chip area and net path length are considered. When undesirable block shapes and pin positions are detected, alternate floorplan templates and room assignments are tried by backtracking

and using automatic module generators. This algorithm performs better than other well-known deterministic algorithms and generates solutions comparable to random-based algorithms.

Ting-Chi Wang and D. F. Wong [7] have presented an optimal algorithm for a special class of floorplans called *hierarchical floorplans of order 5*. Two types of blocks have been considered; L-shaped and rectangular. The algorithm takes a set of implementations for each block as input and identifies the best implementation for each block so that the resulting floorplan has minimum area.

Existing Algorithms for Floorplanning of MBC Designs

All the algorithms discussed in the previous section can be used for floorplanning of MBC designs. However, these algorithms were not implemented as a part of any tool which can generate floorplans for MBC designs. In this section, we describe some of the algorithms that were developed as a part of a tool specifically designed for floorplanning of MBC designs.

In [8], a heuristic algorithm has been developed for MBC designs. The algorithm employs a combined floorplanning, partitioning and global routing strategy. The main focus of the algorithm is in reducing the white space costs and the wiring cost. In [12], the simulated annealing approach is used to solve the floorplanning problem for MBC designs. In [9], "CHAMP", a floorplanning tool for MBC designs using the hierarchical approach has been presented. In [10, 11, 13], the floorplanning problem for MBC designs has been considered. All existing floorplanning algorithms, except [13], for the MBC designs restrict the block shapes to rectangular in order to simplify the problem at hand. Even in [13], only the pre-designed block shapes are considered to be rectilinear and the shapes generated for soft modules are always rectangular with varying aspect ratios. None of the existing floorplanning algorithms for MBC designs take advantage of the flexibility of the standard cell regions.

CHAPTER III

MOTIVATION FOR A NEW FLOORPLANNING ALGORITHM

In this chapter, we present our motivation for developing a new floorplanning algorithm for the MBC design style. We will discuss more about flexible blocks, the reason behind their flexibility and the constraints on the shapes that can be assigned to the flexible blocks.

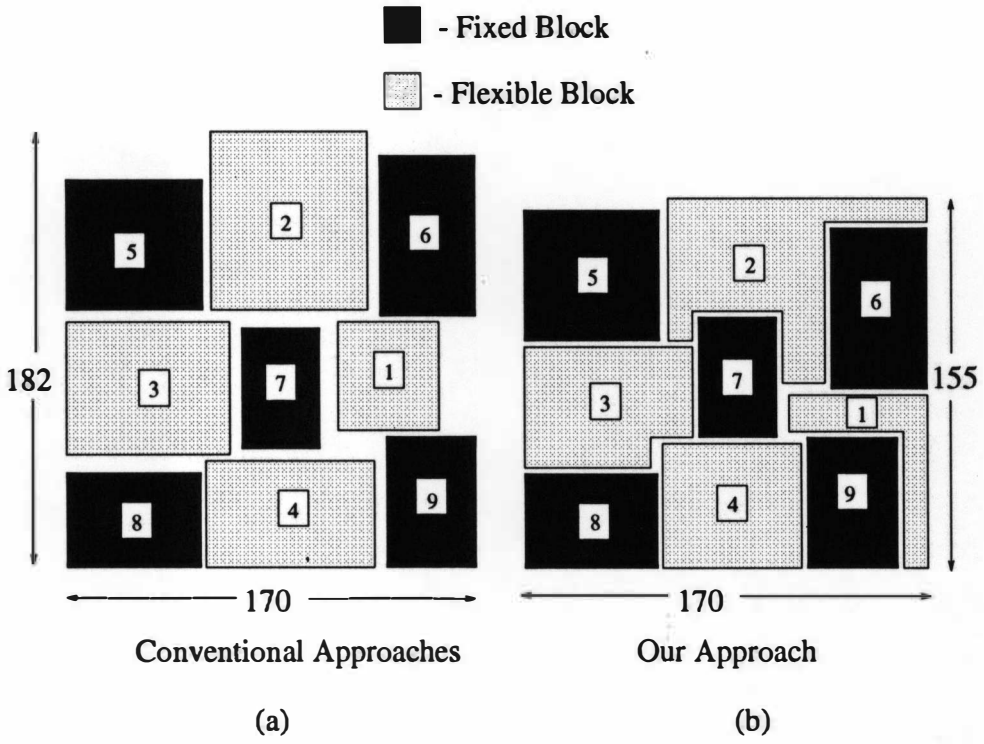

Figure 2. Reduction in Area by Using Rectilinear Shaped Flexible Blocks.

As we saw in the earlier chapter, the main drawback of existing floorplanning algorithms is the inability to utilize the flexibility of the standard cell regions. Figure 2(a), shows a floorplan which is generated by restricting the shapes of flexible blocks to rectangles. Notice that block 2 lies on both the horizontal as well

as the vertical constraint paths. If the shape of block 2 is changed by reducing its height to reduce overall chip height, it will cause an increase in the width of the chip and vice versa. Hence, the only way chip area can be reduced further is to consider rectilinear shaped flexible blocks. As shown in Figure 2(b), by generating rectilinear shaped flexible blocks the area of the floorplan shown in Figure 2(a) has been reduced further. In addition to reduction in area, rectilinear shaped flexible blocks also can cause a reduction in the length of critical paths. As shown in Figure 3, the consideration of rectilinear shapes for standard cell blocks not only leads to reduction in chip area but most importantly it reduces the distances between the terminals of a net thereby improving performance.

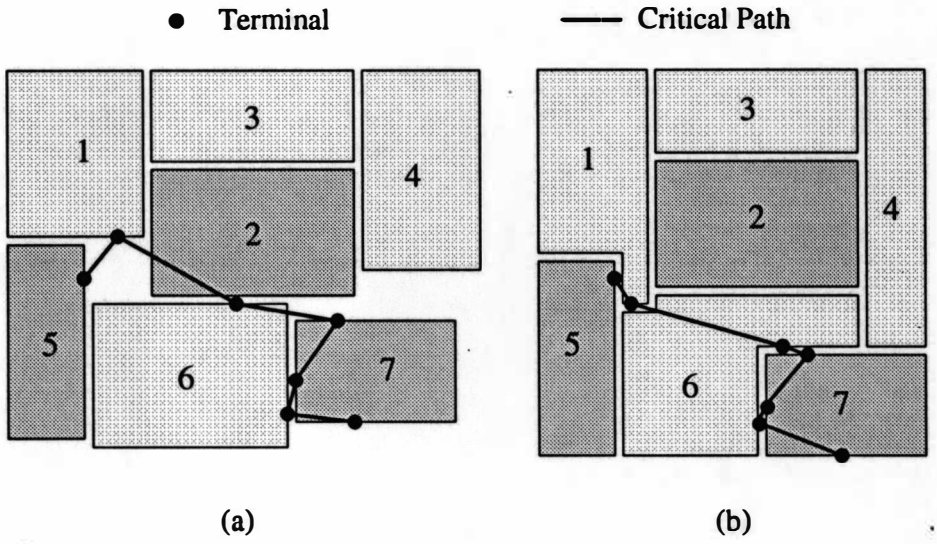

Figure 3. Reduction in Critical Path Lengths by Reshaping Flexible Blocks.

Most of the microprocessor chips designed in the last three years have been developed using the MBC design style. Even though the designer would like to have blocks with rectilinear shapes, the CAD tools available in the industry are not sophisticated enough to handle rectilinear shapes. The main difference between other approaches and our approach is that we can generate rectilinear shapes for flexible blocks. To the best of our knowledge, this work is the first of its kind in

generating truly rectilinear shapes. The most complex shapes considered by other researchers have been limited to L-shapes.

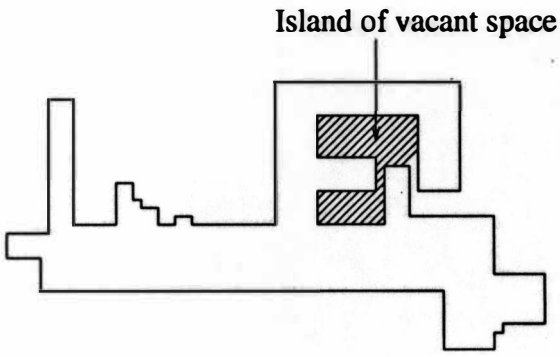

Figure 4. Complex but Undesirable Shape for Flexible Blocks.

Flexible Blocks and Shape Parameters

Algorithms dealing with flexible blocks have to ensure that the shapes of the flexible blocks generated by the algorithm are valid. Generally a block is flexible if it consists of either standard cells or it is block for which only an area estimation is available at the time of Physical Design Automation. In either case, the block shapes generated by the floorplanning algorithm need to be controlled. This is done in order to avoid long thin blocks which may be completely useless. In case of standard cell blocks, the designer would like to restrict the minimum size of the blocks so that a specific number of rows or columns of standard cells can be placed within the block. Traditionally, block shapes have been restricted to rectangular in order to simplify the problem. The shapes of the rectangular blocks can be controlled by using the aspect ratio shape parameter. For each flexible block a range of aspect ratios is specified. The floorplanning algorithm has to generate a shape so that it adheres to the aspect ratio limitation on the block.

In case of ARCHITECT, as we deal with rectilinear shaped regions, the aspect ratio shape parameter cannot be used to restrict the shapes of the flexible blocks. Some complex rectilinear shapes as shown in Figure 4, are completely useless. Some of the regions of the flexible block shown in Figure 4 are too thin for assigning standard cells in those regions. Also, there are "islands" of vacant space which is highly undersirable. In order to control the shape generation of flexible blocks and to avoid undesirable shapes, we have defined two different shape parameters which can control the shapes of these blocks. Each flexible block is represented as a set of connected edges as shown in Figure 5(a). It is obvious that

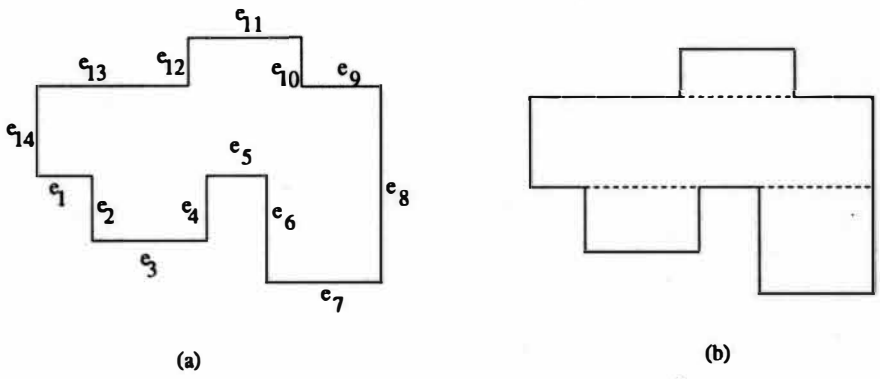

Figure 5. Representation of Flexible Blocks.

we need one shape parameter which will control the number of sides or edges of the flexible block. The shape parameter which controls the number of edges of a flexible block is called α. When $\alpha = 4$, the shapes of the flexible blocks are restricted to rectangular shapes and as the value of α is increased, highly complex shapes can be generated. A rectangular decomposition of a rectilinear region, as shown in Figure 5(b), will yield rectangles of different widths and heights. For these regions to be useful, we must ensure that standard cells can be assigned within that region. It might so happen that some or all of these regions may have dimensions which cannot be used for generating standard cell rows within

these regions. To avoid this, we use another shape parameter, called β which can enforce this limitation on the flexible block. β is the minimum length of each side or edge of a flexible block. It is also the minimum distance between two parallel sides of a flexible block. The requirement on the length of each side is obvious but we also need to have the restriction between two parallel edges of the flexible block in order to avoid shapes as shown in Figure 6. Each flexible block in the design can have its own α shape parameter but usually the β shape parameter is the same for all the flexible blocks.

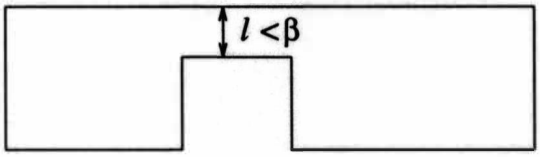

Figure 6. Requirement of β Shape Parameter for Flexible Blocks.

In addition to α and β, we define certain properties that any rectilinear shape must have in order for it to be a valid shape. These properties are motivated by our algorithm. It is obvious that any rectilinear shape will have an even number of edges. Hence the value of α should be an even integer. Apart from this inherent property of the rectilinear blocks, we specify two more properties which any rectilinear region should have for it to be valid. These properties are:

B1: The topmost horizontal edge, or the horizontal edge which has maximum y-coordinate, is the longest horizontal edge among all the horizontal edges of the rectilinear shape.

B2: The length of the topmost horizontal edge is equal to the sum of the lengths of all the other horizontal edges.

Lemma 1 *The aspect ratio range for a flexible block can always be expressed as* $a_{min} \leq a \leq \frac{1}{a_{min}}$, *where,* a *is the aspect ratio of the flexible block and* a_{min} *is the minimum aspect ratio for the flexible block.*

Proof:

It is easy to see that a block having aspect ratio a_{min}, when rotated by 90^o, will have an aspect ratio $\frac{1}{a_{min}}$. Hence, the upper bound on the aspect ratio of a flexible block is an inverse of its lower bound or vice versa.

Lemma 2 *The shape parameters α and β can replace aspect ratios for rectangular flexible blocks.*

Proof:

If aspect ratio is defined as the ratio of height of the block to its width then,

$$a_{min} = \frac{h_{min}}{w_{max}}$$

or

$$w_{max} = \frac{h_{min}}{a_{min}}$$

where, h_{min} is the minimum height of the block and w_{max} is the maximum width possible for the block. Also, we know that if A is the area of the block then,

$$A = h_{min} \times w_{max}$$

$$i.e\ h_{min} = \frac{A}{w_{max}}$$

$$= \frac{A \times a_{min}}{h_{min}}$$

$$h_{min}^2 = A \times a_{min}$$

$$i.e\ h_{min} = \sqrt{A \times a_{min}}$$

Hence, shape parameters α and β act like aspect ratios for rectangular flexible blocks when $\alpha = 4$ and $\beta = \sqrt{A \times a_{min}}$.

CHAPTER IV

AN OVERVIEW OF THE ARCHITECT SYSTEM

In this chapter, we present an overview of our ARCHITECT floorplanning system. Before we present the overview, we describe the terminology that will be used along with the input requirements for ARCHITECT.

Preliminaries

The input to the ARCHITECT system consists of a set of fixed blocks, $\mathcal{B}_p = \{b_1, b_2, \ldots, b_p\}$, a set of standard cells, $\mathcal{S} = \{s_1, s_2, \ldots, s_q\}$, a set of nets, $\mathcal{N}_p = \{n_1, n_2, \ldots, n_r\}$, a set of critical paths $\mathcal{C}_p = \{c_1, c_2, \ldots, c_s\}$ and the target chip width, W and height, H. Also specified for each fixed block b_i, is a three tuple (w_i, h_i, T_i^b), where, w_i is the width, h_i is the height and T_i^b is the terminal information. The terminal information $T_i^b = \{(t_1, x_1, y_1), (t_2, x_2, y_2), \ldots, (t_t, x_t, y_t)\}$, where t_i is a unique terminal number, (x_i, y_i) give the coordinates of terminal, t_i, with respect to the lower left corner of b_i and t is the number of terminals for b_i. A standard cell, $s_i, 1 \leq i \leq q$, is represented by a 2-tuple (w_i, T_i^c), where w_i is the width of s_i and $T_i^c = \{(t_1, x_1, y_1), (t_2, x_2, y_2), \ldots, (t_u, x_u, y_u)\}$, is the set of terminals which belong to s_i, where t_u is a unique terminal number, (x_u, y_u) denote the coordinates of t_u with respect to the lower left corner of the standard cell and u is the number of terminals for s_i. We assume that all the standard cells belong to the same library and hence have the same height. A *net* n_i, $1 \leq i \leq r$, is a wire which connects a group of terminals given by $T_{n_i} = \{t_1, t_2, \ldots, t_v\}$, where each terminal, t_i, either belongs to a fixed block or a standard cell. Each critical path, $c_i = \{a_j | a_j \in \mathcal{B}_p \text{ or } a_j \in \mathcal{S}\}$, is a set of components through which the path passes starting from the input to the output.

Let the length of a net, n_i be given by $l(n_i)$ and that of a critical path be given by $l(c_i)$. The objective of the ARCHITECT system is to identify the locations, (x_i, y_i) for each macro block and standard cell such that the area of the chip, the total wirelength $(\sum_{i=1}^{r} l(n_i))$ and the length of each critical path $(\sum_{i=1}^{s} l(c_i))$ are minimized.

Overview of the ARCHITECT Floorplanning System

The ARCHITECT floorplanning system consists of the following four phases.

1. **Partitioning the Netlist:** The ARCHITECT system iteratively partitions the netlist $\mathcal{N}_p$ till each partition consists either of a single macro block or only standard cells and no macro blocks as is done in [8]. Hence this partitioning process generates a new set of blocks consisting of the original fixed blocks and some flexible blocks.

2. **Initial Placement:** The objective of the Initial Placement (IP) algorithm is to determine the relative positions and orientations for the blocks generated by the partitioning phase. Either a constructive or iterative placement algorithm can be used. We use a simple hierarchical approach to generate the relative placement of the blocks. The objective of our initial placement algorithm is to identify the relative positions of the blocks with respect to one another, reduce netlengths and critical path lengths.

3. **Flexible Block Reshaping:** The floorplan for the MBC design is actually generated in this phase. The input to this phase is an initial placement which consists of both fixed and flexible blocks. The Flexible Block Reshaping (FBR) algorithm not only determines the shapes of the flexible blocks but also compacts the floorplan. It uses the initial placement information to determine the relative positions of the blocks. Hence the initial placement generated does not have to be compact. The FBR algorithm is an iterative algorithm and it compacts the floorplan alternately in the x and y directions. Rectilinear shapes are

generated for flexible blocks in order to minimize the overall area of the floorplan. The FBR algorithm is applied to the floorplan till no significant improvement in the floorplan area is observed. The FBR problem is to determine the shapes of the flexible blocks such that the overall chip area is minimized subject to the constraints that the the relative positions of the blocks are maintained, the overall critical path length generated by the initial placement is not exceeded, both α and β constraints on each block are maintained and the properties **B1** and **B2** for all flexible blocks are not violated.

4. **Standard Cell Assignment:** The standard cell assignment is carried out after the floorplan for the MBC design has been determined. At this stage, the locations of all the blocks are fixed and the shapes of all the flexible blocks have been determined. In this phase, the placement of standard cells within each flexible block is carried out. The channel estimation between standard cell rows as well as the creation of feedthroughs is accomplished in this phase. We use the SOAP [14] algorithm to assign standard cells within the flexible blocks.

CHAPTER V

THE ARCHITECT FLOORPLANNING SYSTEM

In this chapter we describe our Flexible Block Reshaping algorithm for floorplanning of MBC designs. For the sake of clarity and continuity, we describe each phase of ARCHITECT. However, the emphasis of this chapter is on the FBR algorithm which is the most important phase of ARCHITECT. It is in this phase, that we generate a floorplan for MBC designs with rectilinear shaped flexible blocks.

Partitioning the Netlist

The ARCHITECT system iteratively partitions the netlist $\mathcal{N}_p$, till each partition consists either of a single macro block or only standard cells and no macro blocks as in [8]. Any partitioning algorithm which satisfies this constraint can be used. Hence the partitioning process generates a new set of blocks, $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, consisting of macro blocks and flexible blocks. If a block B_i, is a macro block then it is represented by a five tuple $(x_i, y_i, o_i, w_i, h_i)$, where, (x_i, y_i) represent the coordinates of the lower left corner of the block, o_i determines the orientation of the block, w_i represents the width and h_i represents the height of the block. If the block B_i, is a flexible block then it is represented by a five tuple $(x_i, y_i, o_i, S_i, E_i)$, where, (x_i, y_i) are the coordinates of the origin point of the flexible block which is initially the lower left corner of the block. The *origin* is defined as a point which has the minimum y-coordinate among the set of points which have minimum x-coordinate. o_i determines the orientation of the flexible block before it is reshaped. S_i is the set of standard cells contained within the flexible block, and $E_i = \{e_1^i, e_2^i, \ldots, e_n^i\}$ gives the sequence of line segments which define the boundary of the flexible block starting from the

origin point and proceeding in an anticlockwise direction. Partitioning the netlist $\mathcal{N}_p$, gives rise to a new netlist, $\mathcal{N} = \{N_1, N_2, \ldots, N_m\}$. Each net N_i, is a set of terminals of the blocks of $\mathcal{B}$, which are to be connected by a wire. Also, a new critical path list, $\mathcal{C} = \{C_1, C_2, \ldots, C_o\}$ is constructed, where each critical path $C_i = \{a_1, a_2, a_3, \ldots, a_q\}$, where each $a_j \in \mathcal{B}, 1 \leq j \leq q$.

The Initial Placement Phase

The main objective of the Initial Placement phase is to generate a relative placement of blocks. The objective does not include generation of a minimum area placement for the blocks. We use a hierarchical approach to generate the initial placement. If $l(N_i)$ denotes the length of a net N_i and $l(C_j)$ denotes the length of the critical path C_j, then our Initial Placement (IP) algorithm determines the coordinates (x_i, y_i) and orientation, o_i, for each block $B_i \in \mathcal{B}$ subject to the constraints that both $\sum_{i=1}^{m} l(N_i)$ and $\sum_{i=1}^{o} l(C_i)$ are minimized. The generation of initial placement is carried out in three steps which are described below.

1. **Cluster tree formation:** The first step is to generate a physical hierarchy or the cluster tree $\mathcal{T}$. The cluster tree $\mathcal{T}$, is generated by clustering the blocks $\mathcal{B}$, in a bottom-up fashion. Each leaf node of the tree represents a block $B_i \in \mathcal{B}$, where $i \leq i \leq n$. Hence, the number of leaf nodes is equal to n. The leaves represent the level 0 of the tree and the root of the tree represents the highest level of the tree. At each level, l, of the hierarchy, blocks (or clusters) which are highly connected are grouped together. Cluster $C_{i,j}$ represents a group of blocks at node j in level i of $\mathcal{T}$. If $\mid C_{i,j} \mid$ represents the size of the cluster then $\mid C_{1,j} \mid \leq 4$, and $\mid C_{i,j} \mid \leq 2$, where $i \geq 2$. The reason for doing this is to simplify the search procedure used later.

2. **Relative positioning of the clusters in the cluster tree:** The cluster tree, $\mathcal{T}$, generated in the previous step identifies the clusters that would be placed close together in a floorplan but their relative positions are not yet

decided. In this step, relative positions of all the clusters are determined with respect to one another. The root of $\mathcal{T}$ represents the floorplan of the entire chip. The target aspect ratio of the chip sets the target shape for the root node of $\mathcal{T}$. In this step, a top-down traversal of $\mathcal{T}$ is performed. For distribution of aspect ratios, we consider two templates, one which is split by a horizontal cutline and the other which is split by a vertical cutline, for possible room assignments at each level. We need to consider only two templates as each cluster $\mid C_{i,j} \mid \leq 2, i > 1, \forall j$. This phase creates constraints on the aspect ratios of the rooms which are recursively transmitted down the cluster tree $\mathcal{T}$. Hence, the aspect ratios of the clusters of $C_{l,j}$ $\forall l, j$ are determined from the aspect ratio of the respective parent clusters $C_{l+1,k}$ $\forall k$. The traversal of $\mathcal{T}$ is stopped when $C_{1,j}$ $\forall j$ have been assigned a target aspect ratio. While traversing down $\mathcal{T}$, in addition to the aspect ratio assignment, the global routing information is also transmitted in terms of I/O pad assignment to all of the clusters. Given an overall aspect ratio goal and I/O pad goal, at each level, l, of the hierarchy, the floorplanner generates the aspect ratio goals and I/O pad goals for $C_{l,j}$ $\forall j$.

3. **Relative positioning of blocks in a cluster:** In this step we determine the relative positions of the blocks in the cluster at level 1. Since all the clusters at level 1 have a target aspect ratio and an I/O pad goal assigned to them in the previous step, in this step, we position all the blocks and fix their orientations within the cluster so that the target aspect ratio and I/O pad goal is achieved. Since there are at most four blocks within the cluster at this level, we can carry out an exhaustive search.

Flexible Block Reshaping Algorithm

In this section, we describe our FBR algorithm for floorplanning of MBC designs. The FBR algorithm improves the initial layout generated by the initial placement algorithm by assigning rectilinear shapes to flexible blocks. The

floorplan is improved by using a one dimensional compaction combined with reshaping of flexible blocks which reduces the white space in the floorplan. The algorithm is iterative in nature. The floorplan is divided into several rows of blocks. The blocks are processed one row at a time and are added to the already processed set of blocks. Hence, we have two sets of blocks; one is a processed set of blocks which is also called a *partial floorplan* and the other set consists of unprocessed blocks. The top boundary of the partial floorplan is a set of line

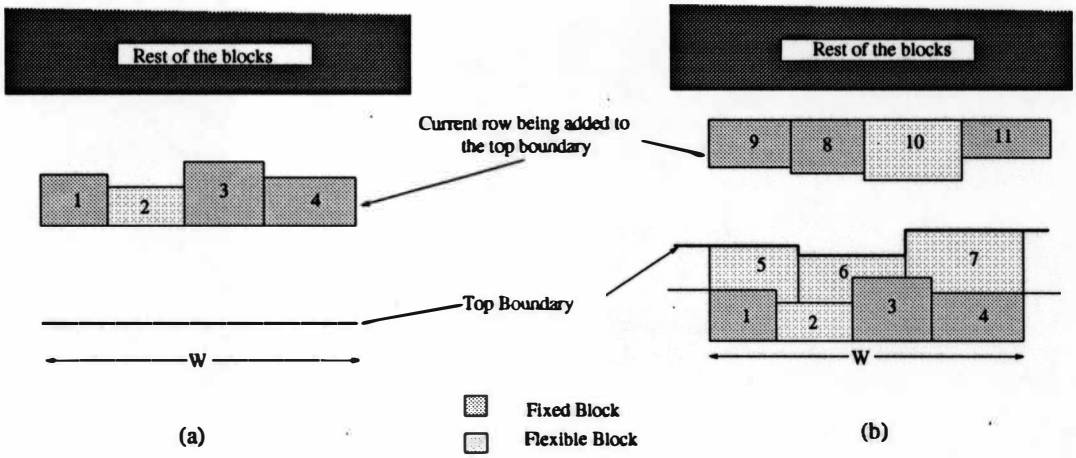

Figure 7. The FBR Process.

segments, $L = \{l_1, l_2, \ldots, l_t\}$. Each line segment l_i, is stored by its two end points, i.e. $l_i = \{(x_i^1, y_i^1), (x_i^2, y_i^2)\}$. Initially when none of the blocks are processed, the top boundary is a straight line as shown in Figure 7(a). After a set of blocks are processed and added to the partial floorplan the top boundary is modified according to the blocks that were added to the top boundary as shown in Figure 7(b). In a single iteration, all the blocks are processed and added to the partial floorplan. After all the blocks are processed, the floorplan is rotated and the process is repeated in the other direction. This process is repeated until there is no significant improvement in the floorplan area.

Each iteration of the FBR algorithm consists of three steps. In the first step, the set of processed blocks is empty and the top boundary is a straight line. Hence, when the first row of blocks is being processed, it has to be assigned to a straight boundary as shown in Figure 7(a). Hence, this problem is called the Straight Boundary Block Assignment (SBBA) problem. After assigning a row of blocks to the top boundary, the top boundary is no more a straight line but it consists of a set of vertical and horizontal line segments as shown in Figure 7(b). The next row of blocks being processed has to be assigned to this irregular rectilinear top boundary. Hence, this problem is called the Irregular Rectilinear Boundary Block Assignment (IRBBA) problem. All further rows of blocks except the last row can be assigned to the top boundary by using the IRBBA phase of the algorithm. The last row of blocks has not only to be assigned to the irregular rectilinear top boundary but also the new top boundary generated should be as straight as possible. We call this problem the IRBBA problem with the Straight Top or IRBBAST problem. Each of the phases of FBR is described in detail in the following sections.

The line segments in L have a type associated with them. The *type* of a line segment is defined as,

$type(l_i) = h$, if the line segment is horizontal

$type(l_i) = v$, if the line segment is vertical.

The line segments in L satisfy the following properties:

P1: $type(l_i) \neq type(l_{i+1})$, $1 \leq i \leq t-1$.

P2: $l_i \cap l_j = \phi$, $1 \leq i,j \leq t$ where the line segments l_i and l_j are horizontal.

P3: The first and last line segments of the top boundary are always horizontal and number of horizontal line segments will be one more than number of vertical line segments.

P4: If $l(l_i)$ denotes the length of line segment $l_i \in L$, then $l(l_i) = \beta$, if $type(l_i) = h, \forall l_i \in L$.

A vertical line segment which has an end point with the y-coordinate greater(less) than that of the immediate preceding horizontal line segment in L, is called a *rising vertical* (*falling vertical*) line segment and is said to be a v_r (v_f) segment. If a line segment $l_i \in L$ is a v_f line segment and the line segment l_{i+2} is a v_r line segment, then the line segments l_i, l_{i+1} and l_{i+2} are said to define a *depression* in the top boundary of the partial floorplan as shown in Figure 8.

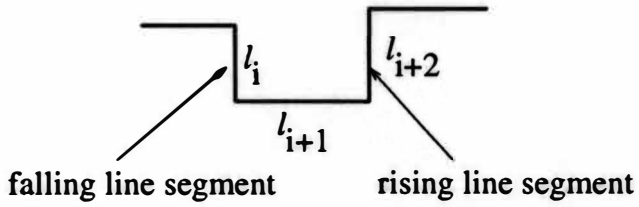

Figure 8. A Depression in the Top Boundary.

Straight Boundary Block Assignment

Initially, when none of the blocks of the floorplan have been processed, the partial floorplan does not consist of any block and the partial floorplan boundary is a straight line segment of width W. We process the first row of blocks together so as to assign them within width W on the partial floorplan boundary. To solve the SBBA problem, we restrict the value of α to 4 for all the flexible blocks i.e. all the flexible blocks are restricted to rectangular shapes only. The objective of this problem is to assign a set of blocks to a straight top boundary such that the difference in heights of the tallest flexible block and the shortest flexible block after assignment to the top boundary is minimized. This can be done by reshaping some or all of the flexible blocks. Let $\mathcal{B}_\pi = \{B_1, B_2^*, B_3, B_4^* \ldots, B_n\}$, be the set of rectangular blocks such that, $B_i^* \in \mathcal{B}_\pi$ is a flexible block and $B_i \in \mathcal{B}_\pi$ is a fixed

block. Each block B_i (B_i^*) has an area A_i (A_i^*), width w_i (w_i^*) and height h_i (h_i^*) and $\sum_{\forall B_i \in \mathcal{B}_\pi} w_i + \sum_{\forall B_i^* \in \mathcal{B}_\pi} w_i^* = W'$. Let $B_f \subseteq \mathcal{B}_\pi$, be the set of flexible blocks in $\mathcal{B}_\pi$ and let $\mid B_f \mid = r$. We define $\delta = max\{h_i^*\} - min\{h_i^*\}$ where $max\{h_i^*\}$ is the height of the tallest flexible block and $min\{h_i^*\}$ is the height of the shortest flexible block in $\mathcal{B}_\pi$.

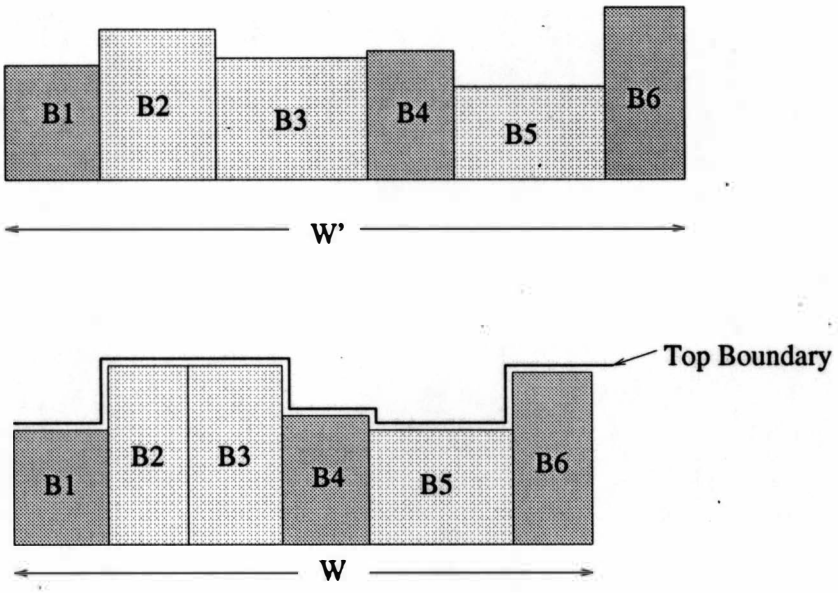

Figure 9. The SBBA Problem and Its Solution.

In SBBA, we are given,

1. Permutation $\mathcal{B}_\pi$,

2. $W \leq W'$, the total width within which the blocks must be assigned to the partial floorplan boundary.

We have to assign the blocks in $\mathcal{B}_\pi$ to the horizontal top boundary such that $\sum_{\forall B_i^* \in \mathcal{B}_\pi} w_i^* + \sum_{\forall B_i \in \mathcal{B}_\pi} w_i = W$, δ is minimum, the given permutation of blocks is not altered, the value of α is restricted to 4 for all flexible blocks and β constraint is not violated for any of the flexible blocks. A SBBA problem is shown if Figure 9(a) and its solution is shown in Figure 9(b). For the sake of clarity, the top boundary in Figure 9(b) has been displaced from the blocks.

It is clear that an assignment of all the blocks in $\mathcal{B}_\pi$ to the top boundary such that the total width of the row is W is possible only if

$$r\beta + \sum_{\forall B_i \in \mathcal{B}_\pi} w_i \leq W$$

Each flexible block in the set B_f is initially assigned a width β i.e.

$$w_i^* = \beta, \quad \forall B_i^* \in B_f$$

The height of each block can then be calculated as:

$$h_i^* = \frac{A_i^*}{\beta}, \quad \forall B_i^* \in B_f$$

Since we have assigned the minimum possible width to the flexible blocks, there is some excess width W_e which can be assigned to the flexible blocks. This excess width is given by:

$$W_e = W - \sum_{\forall B_i \in \mathcal{B}_\pi} w_i + W_f$$

where,

$$W_f = \sum_{\forall B_i^* \in \mathcal{B}_\pi} w_i^*$$

Initially when all the flexible blocks are assigned a width β, $W_f = r\beta$. The widths of the flexible blocks have to be increased such that, $W_e = 0$, δ is minimized and β constraint is not violated for any of the flexible blocks. We sort the blocks in $\mathcal{B}_f$ in the non-descending order of their heights. When all the flexible blocks have the same height, $\delta = 0$. This is the lower bound on the value of δ and it may not be possible to achieve this lower bound because of the β constraint on the flexible blocks. Our solution consists of two steps. In the first step, we identify a flexible block B_p^* from the sorted list, with height h_p^* such that all other flexible blocks $B_j^* \in B_f, j > p$ can be assigned height h_p^* and the new width W_f, of all the flexible blocks satisfies:

$$W_f \leq W - \sum_{\forall B_i \in \mathcal{B}_\pi} w_i$$

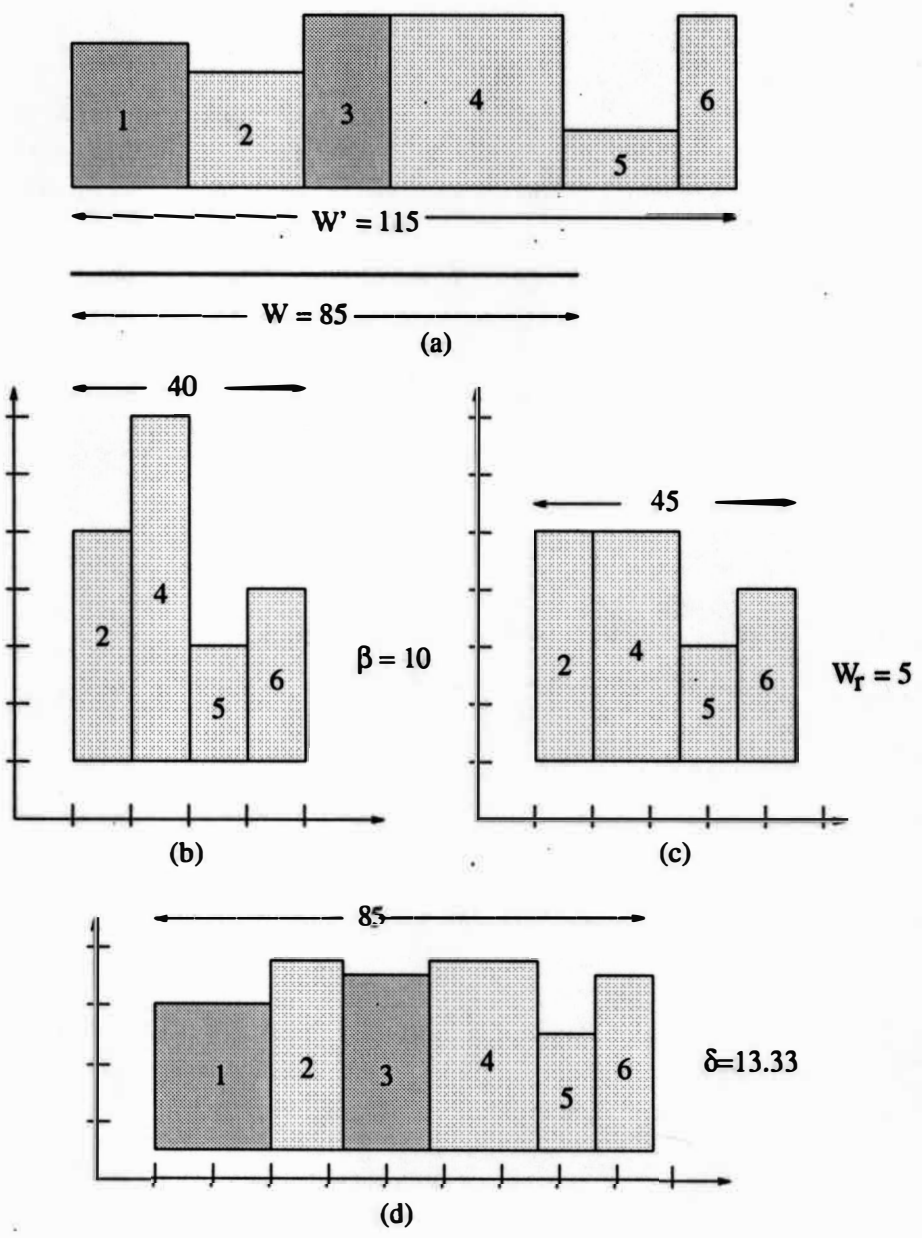

Figure 10. The SBBA Solution for an Example.

The block B_p^* identified is such that, if all blocks with height greater than that of block B_{p+1}^* are assigned height h_{p+1}^* then

$$W - \sum_{\forall B_i \in \mathcal{B}_*} w_i - W_f \leq 0$$

The search for block B_p^* is carried out using a binary search among the sorted list of flexible blocks. Let $W_r = W - \sum_{\forall B_i \in \mathcal{B}_*} w_i - W_f$. In the second step, this widths of all the flexible blocks, B_q^* where $q \geq p$, is increased so that $W_r = 0$. The new height of the flexible blocks B_q^*, $q \geq p$ is calculated as:

$$h_q^* = \frac{\sum_{i=q}^{r} A_i}{\sum_{i=q}^{r} w_i^* + W_r}$$

Using this new height, we recalculate the width of the flexible blocks, B_i^*, $p \leq i \leq r$.

Theorem 1 *SBBA problem can be solved in $O(r \log r)$ time.*

The sorting of the blocks according to their heights requires $O(r \log r)$ time and the search for the block in the first step requires $O(r \log r)$ time. The change in widths and height of all blocks $B_i^*, p \leq i \leq r$ requires $O(r)$ time. The second step, in which the remaining width W_r is distributed, requires $O(r)$ time. Hence, the enitre solution for the SBBA problem can be computed in $O(r \log r)$ time. Figure 10 shows the SBBA solution for an example.

In Figure 10(a) shows the input to the SBBA problem; there are two fixed blocks (1 & 3) and four flexible blocks (2, 4, 5 & 6). $W = 85$, $W' = 115$ and $\beta = 10$. We assign the width of all flexible blocks to β as shown in Figure 10(b). Next we identify block 2 as block whose height should be assigned to all blocks taller than block 2. This height assignment is shown in Figure 10(c). Finally the remaining width W_r, is assigned to blocks 2 and 4 and the solution to the SBBA problem is shown in Figure 10(d).

Irregular Rectilinear Boundary Block Assignment

After the first row of blocks has been processed, the top boundary of the partial floorplan may not be a straight line. It may consist of a set of horizontal and vertical line segments. Hence, the problem in this case is to reshape and assign the blocks to an irregular rectilinear boundary with total width W, such that the flexible blocks do not violate properties **B1** or **B2** and the α and β constraints imposed on them. Ideally, we would like to reduce the height of the the resulting partial floorplan. These complex and conflicting objectives make this problem very hard. Hence we make use of heuristics to solve this problem.

In IRBBA, we are given,

1. Permutation $\mathcal{B}_\pi$,

2. $W \leq W'$, the total width within which the blocks must be assigned to the partial floorplan boundary.

3. L, the top boundary of the partial floorplan.

We have to obtain a placement of blocks in $\mathcal{B}_\pi$ on the top boundary such that $\sum_{\forall B_i^* \in \mathcal{B}_\pi} w_i^* + \sum_{\forall B_i \in \mathcal{B}_\pi} w_i = W$, the given permutation of blocks is not altered, the value of α and β is not violated for any of the flexible blocks and all flexible blocks retain their properties **B1** and **B2**. An IRBBA problem and its solution is shown in Figure 11.

Our heuristic solution to the IRRBA problem consists of two steps. In the first step, we solve the SBBA problem for the blocks in $\mathcal{B}_\pi$ as shown in Figure 11(b). This ensures $\sum_{\forall B_i^* \in \mathcal{B}_\pi} w_i^* + \sum_{\forall B_i \in \mathcal{B}_\pi} w_i = W$. In addition, we now have a target range for each block on the top boundary as shown in Figure 11(b) by the dotted lines. The target range R_i for a block $B_i \in \mathcal{B}_\pi$ is given as,

$$R_i = (x_i, x_i + w_i)$$

The target range identifies the x-coordinate range for a block to be placed on the top boundary. Solving the SBBA problem identifies the x_i coordinate of the origin

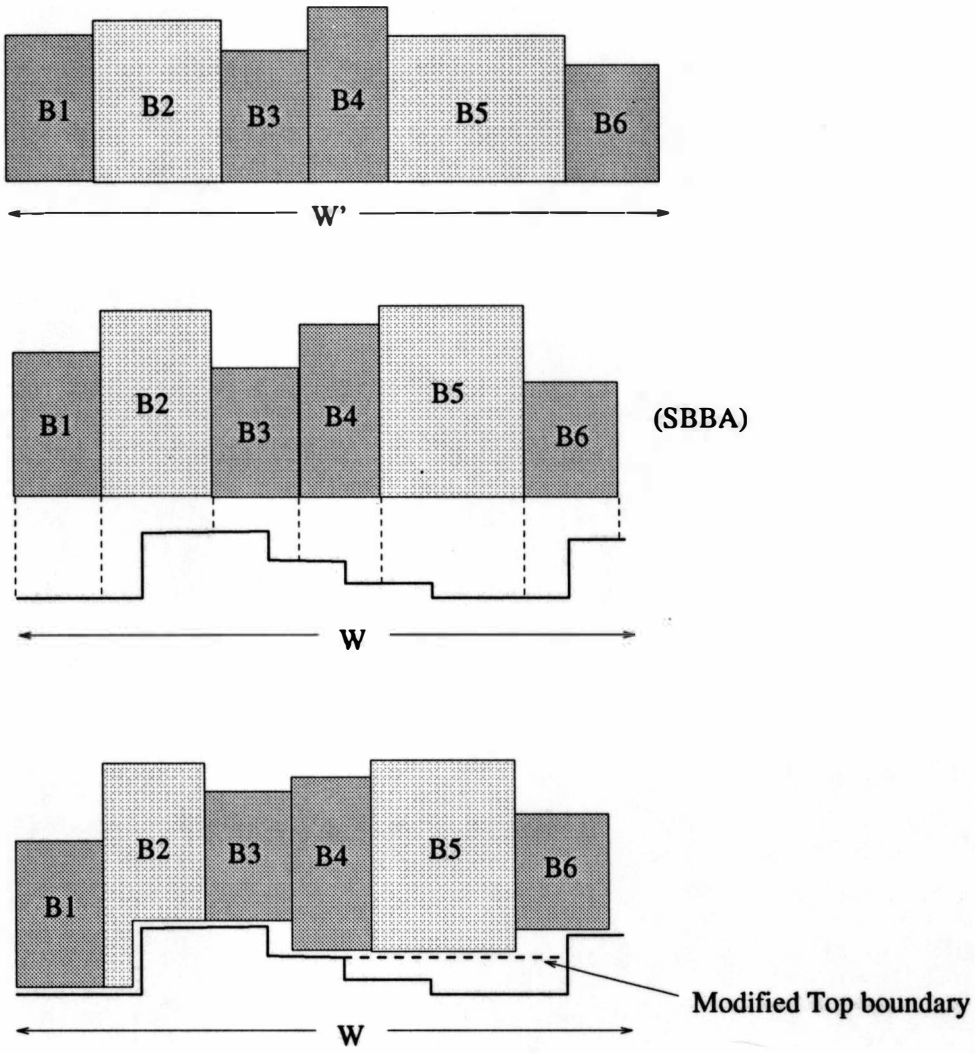

Figure 11. The IRBBA Problem and Its Solution.

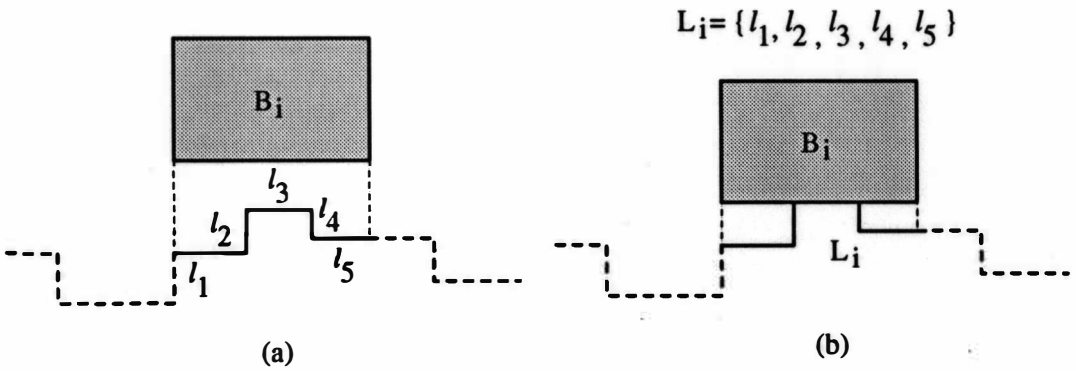

Figure 12. Assignment of a Fixed Block to the Top Boundary.

point for each block in $\mathcal{B}_\pi$. In the second step, we assign each block to its target range on the top boundary. Blocks are assigned to the top boundary one at a time. Before a block, B_i, is assigned, a set of edges, $L_i \subseteq L$ is generated. This set L_i, consists of line segments of the top boundary which have their x-coordinates between x_i and $x_i + w_i$. If the block being assigned to the top boundary is a fixed block, the set L_i is scanned to identify a line segment l_i, such that $type(l_i) = h$ and y-coordinate for this line segment is maximum among all the line segments in L_i. The y-coordinate of the origin point is then set to the y-coordinate of l_i. Figure 12 shows the assignment process for a fixed block. In Figure 12(a), the set of line segments L_i, within the target range for block B_i are shown by continuous lines. The dotted line indicates the rest of the top boundary. The final assignment of the fixed block B_i to the top boundary is shown in Figure 12(b).

When the block to be assigned to the top boundary is a flexible block, the line segments in L_i, determine the shape of the block. The maximum number of line segments the flexible block can have after reshaping is $| L_i | + 3$. This is due to the fact that the bottom of the block gets the shape defined by line segments in L_i and then three additional edges are required to generate a complete or closed region; two vertical segments one with x-coordinate x_i and the other with

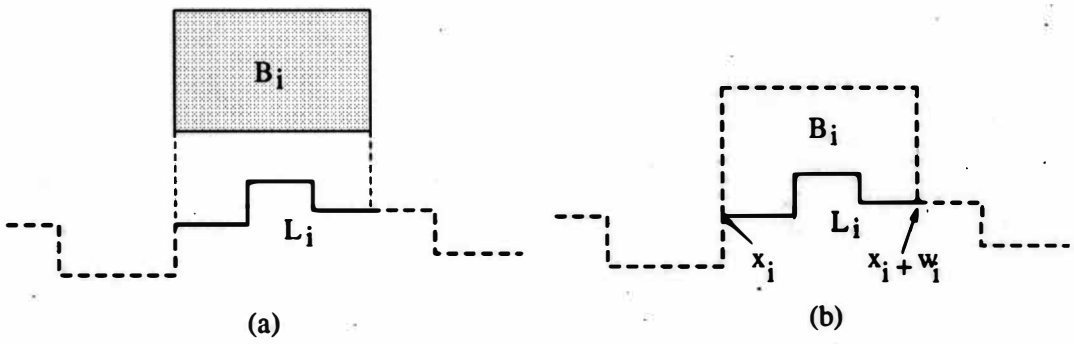

Figure 13. Reshaping of a Flexible Block.

x-coordinate $x_i + w_i$ and a horizontal segment joining these two vertical segments at the top as shown in Figure 13(b). To avoid violation of the α constraint on the block, the following equation must be satisfied:

$$\alpha - \mid L_i \mid -3 \geq 0$$

If the above equation is not satisfied, we identify a depression in L_i which when eliminated will cause the generation of the least amount of white space in the floorplan and modify L_i accordingly. The elimination of a depression reduces the number of line segments in L_i but increases the white space in the layout. This

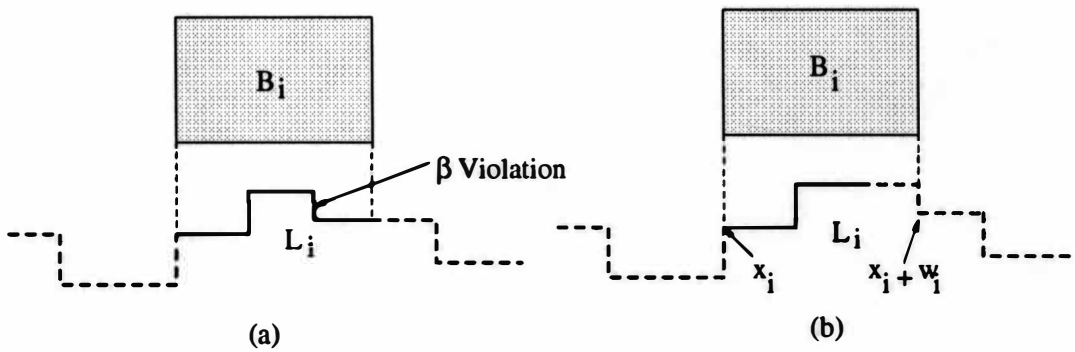

Figure 14. Elimination of Line Segments Violating the β Constraint.

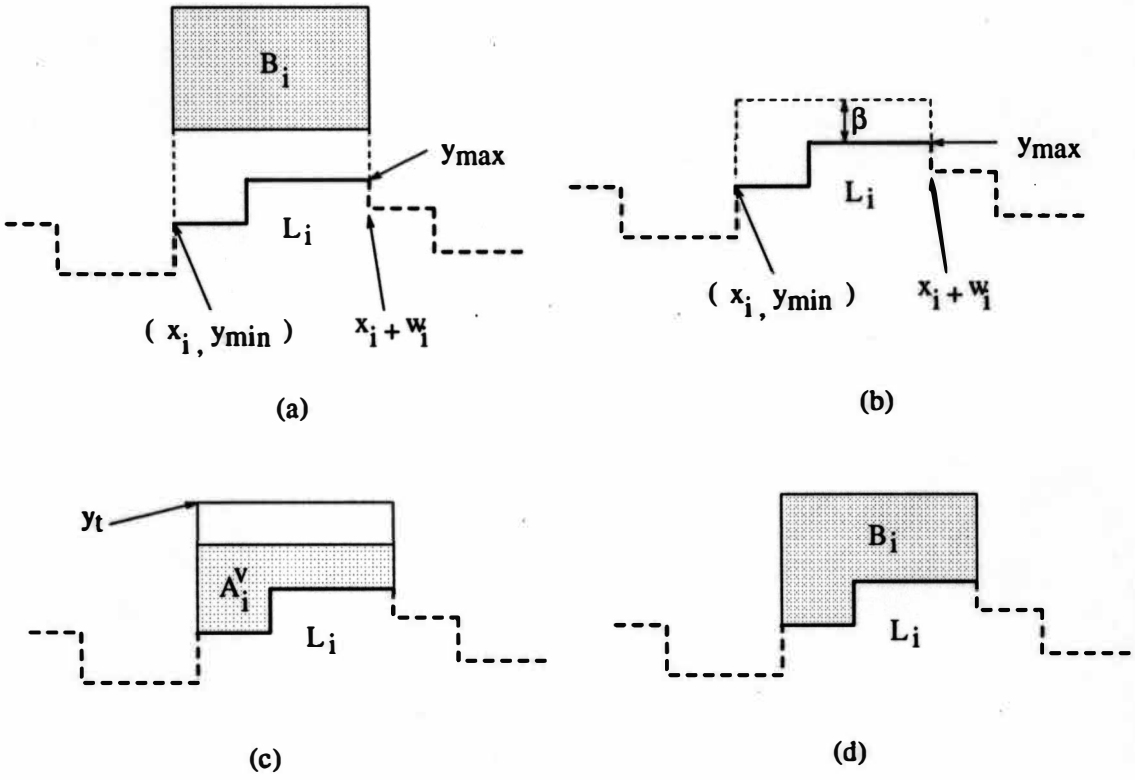

Figure 15. Shape Generation of a Flexible Block.

process is carried out till the above equation is satisfied. The edges in L_i are now scanned for β violations if any. Only the vertical line segments need to be checked as the horizontal segments will always have a length greater than β. The elimination of line segments which violate β constraint on the block is shown in Figure 14. The modified L_i, determines the shape of the bottom of the flexible block. Figure 15(a) shows a block B_i and the new bottom shape determined by L_i for B_i. To complete the reshaping process we need to identify the y-coordinate of the top horizontal edge of the block. For this we use a horizontal scanline of length equal to the target range and scan the region between x_i and $x_i + w_i$ computing the vacant area. Let y_{min} (y_{max}) be the minimum (maximum) y-coordinate of a line segment in L_i. We compute the vacant area, A_i^v, above the top boundary,

between $\{(x_i, y_{min}), (x_i + w_i, y_{min}\}$ and $\{(x_i, y_{max} + \beta), (x_i + w_i, y_{max} + \beta)\}$ as shown in Figure 15(b). If $A_i^v > A_i^*$, then we eliminate more depressions in L_i till $A_i^v \leq A_i^*$. The y-coordinate of the top edge, y_t, of the block is then determined by using,

$$y_t = y_{max} + \beta + \frac{A_i^* - A_i^v}{w_i}$$

Figure 15(c) shows the process of determining the position of the top edge of the

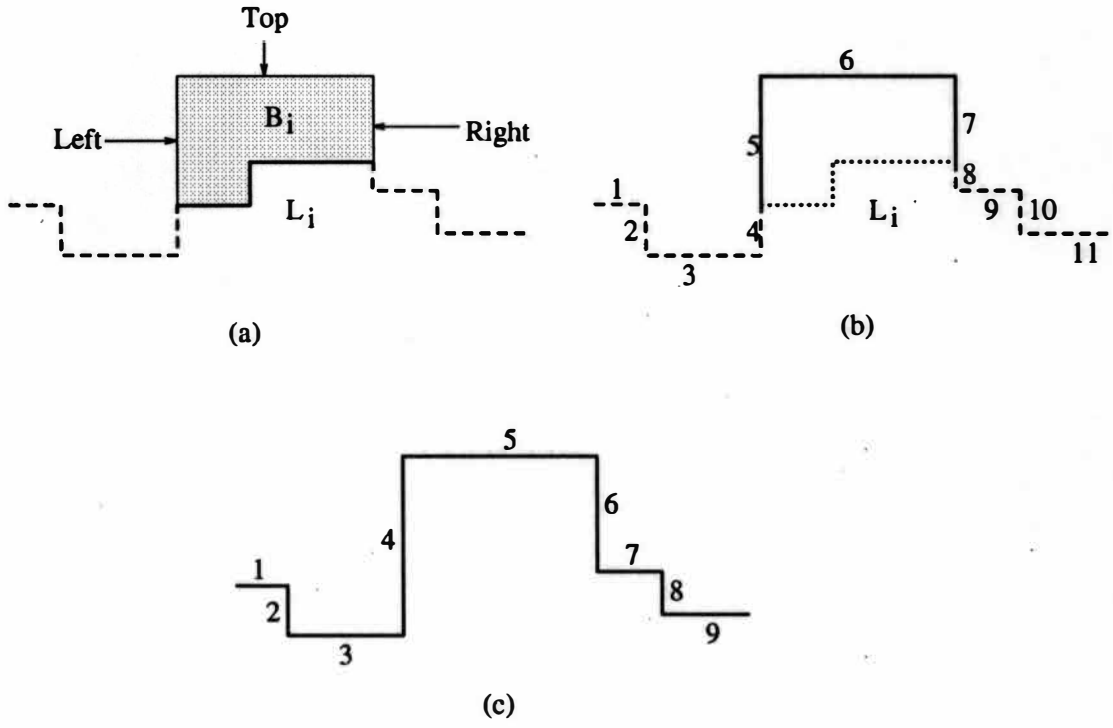

Figure 16. Generation of New Top Boundary After Block Assignment.

flexible block. The top edge of block, either fixed or flexible, is the horizontal edge which has the highest y-coordinate among all the horizontal edges of the block. Figure 15(d) shows the final shape of the block B_i.

After a block has been assigned to the top boundary, we need to modify the top boundary. In order to do that, we eliminate the edges in L_i from the top boundary and add the left, top and right edge of the blocks in that order to

the top boundary as shown in Figure 16(b). Since we also add the left and the right edges, there can be two overlapping vertical edges in L. Also, property **P1** maybe violated. In Figure 16(b), edges 4, 5 and edges 7,8 are two pairs of adjacent vertical edges. Hence they violate the property **P1**. The edges in the new top boundary are scanned so as to eliminate any overlaps and to check that properties **P1**, **P2**, **P3** and **P4** hold true for the new top boundary. When two overlapping edges exist, only the non-overlapping part of the edges is retained in L. In case property **P1** is violated, the two adjacent horizontal or vertical edges are merged to form one edge. The final top boundary generated is shown in Figure 16(c).

Irregular Rectilinear Boundary Block Assignment with Straight Top

The problem of assigning the last row of blocks which form the top boundary of the floorplan is slightly different from the IRBBA problem. This is because, after assigning the blocks to the top boundary, the new top boundary generated must as straight as possible. Hence the problem of assigning the last row of blocks is to place the set of blocks such that the increase in height of the chip is minimized and the bounding rectangle for these blocks has a width no more than W.

In IRBBAST we are given,

1. A permutation $\mathcal{B}_\pi$,

2. W, the total width within which the blocks must be assigned to the partial floorplan boundary.

3. L, the top boundary of the partial floorplan.

The objective is to assign the blocks to the partial floorplan boundary such that, increase in height of the chip is minimum subject to the constraint that the bounding rectangle enclosing all the blocks does not have a width greater than W, the permutation of the blocks given by $\mathcal{B}_\pi$ is not altered, the α and β constraints for the flexible blocks are not violated and all flexible blocks retain the properties **B1** and **B2**.

Figure 17(a) shows an instance of the IRBBAST problem. This problem is a much harder problem compared to the IRBBA problem. Hence, we use a heuristic approach to solve this problem. Our solution consists of two steps. In the first step, we solve the IRBBA problem, as shown in Figure 17(b). Since the top boundary resulting after solving the IRBBA problem is irregular, there are one or more blocks in $\mathcal{B}_\pi$ which define the height of the overall floorplan. If all these blocks are fixed, then a new iteration is started to try and achieve a more compact chip. But if the block(s) are flexible then the next step is to reshape these blocks by assigning these flexible blocks a new target range in order to reduce the overall chip height.

Let, y_t^i denote the y-coordinate of the top edge of a block B_i. Let, B_q be a fixed block in $\mathcal{B}_\pi$ such that:

$$y_t^q = max\{y_t^j\},\ \forall\ B_j\ \in\ \mathcal{B}_\pi$$

We form a set, $\mathcal{B}_f$, of flexible blocks such that:

$$B_j^* \in \mathcal{B}_f,\ \ if\ \ y_t^j \geq y_t^q$$

In Figure 17(b), the tallest fixed block is the block $B4$. Both flexible blocks, $B2$ and $B5$ have y-coordinates of their top edges greater than that of block $B4$. Hence both these blocks will be selected for reshaping. The blocks in $\mathcal{B}_f$ are sorted in the non-ascending order of the y-coordinates of their top edges. Since the top edge of a block is represented by storing the coordinates of its end points, the top edge of each block is of the form $\{(x_{t1}^i, y_t^i), (x_{t2}^i, y_t^i)\}$. Let, B_k and B_l be the two blocks either fixed or flexible that are adjacent to $B_i^* \in B_f$, in the given permutation $\mathcal{B}_\pi$, such that $k < i < l$. The flexible block being processed is removed from its current position on the top boundary and the top boundary is updated. The range R_i, for the flexible block B_i^*, is changed to (x_{t1}^m, x_{t2}^i), where $m = k$ if $y_t^k < y_t^i$ else $m = l$, if $y_t^k > y_t^i$ and $y_t^l < y_t^i$. In the Figure 17(b), the block $B2$ has space on its left as well as to its right. But since we give precedence to the space on the left, the

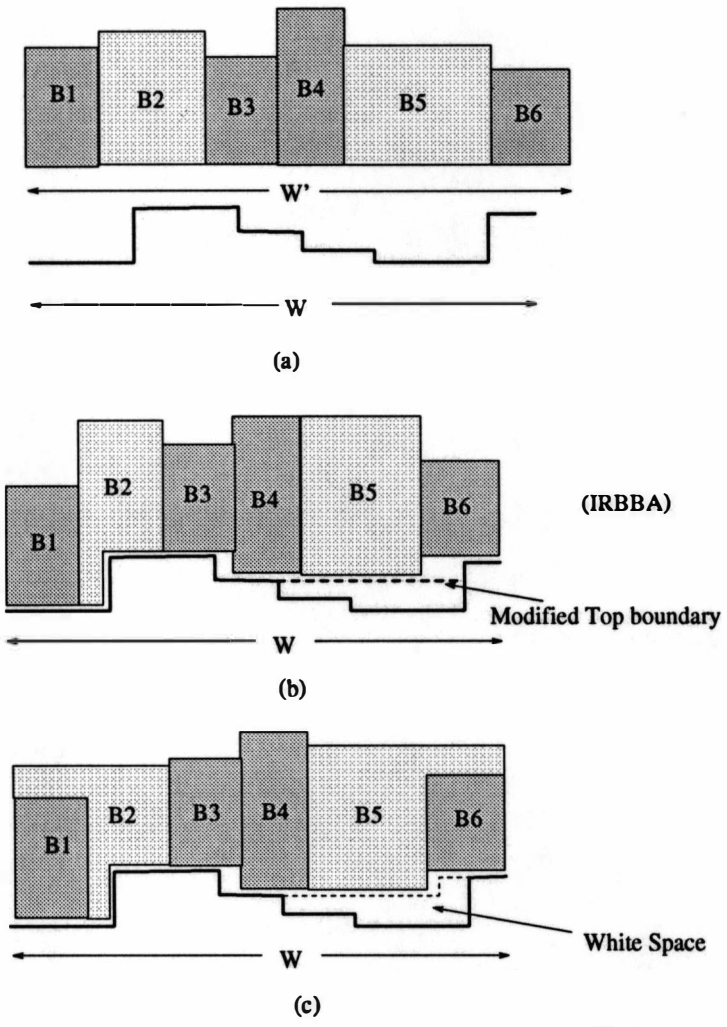

Figure 17. The IRBBAST Problem and Its Solution.

range of the block $B2$ is extended on the left. The block $B5$ has space only on its right. Hence, the range of $B5$ is extended on the right. Using this new range, the block is reshaped as described earlier. This process is carried out for each block in B_f. Figure 17(c) shows the final assignment of shapes for the flexible blocks.

Standard Cell Assignment

At this stage, the floorplan for the MBC design has been generated; the locations of all the blocks have been determined and the shapes of all the flexible blocks are defined. In this phase, ARCHITECT generates the placement of the standard cells within the flexible blocks. This phase is carried out in two steps. The first step is to fix the locations of standard cells which are connected by a critical path or are connected by a net interconnecting different blocks. For a fixed block the position of the terminal of the net is also fixed. But in case of flexible blocks, since the standard cells have not yet been placed, the terminals are not fixed or are *floating*. Hence, for a given net or critical path, we have two types of points; fixed points which represent terminals of a fixed block and floating points which are actually terminals of a standard cell. The problem is to identify a shortest path between these points, some of which are fixed and some of which are floating within a rectilinear boundary. This problem of identifying the location of such cells is similar to the Steiner Minimum Tree problem which is known to be NP-Complete [15]. Hence, we use a heuristic approach to solve this problem. The locations for these cells are generated by using a modified Minimum Cost Spanning Tree algorithm. For a cell in a flexible block, connected by a critical path or a net, the corner points of the flexible block are identified as possible target points to place the cells. That is, the floating terminals within a flexible block are restricted to the corners of the flexible blocks. Hence, for a given critical path C_i, or a net N_i, we have a set of points P_i, some of which are fixed and some of which are floating. To pre-place the cells on this critical path or net,

we construct a complete graph of all the points in P_i. Next, we find a minimum cost spanning tree for these points such that only one corner point of each flexible block lies on the tree. The corner points for a flexible block which lie on the spanning tree are used to pre-place the standard cells. Notice that some of the cells in a region may overlap due to this strategy. But these locations identified for the cells are approximate locations and are improved upon in the next step. The actual placement of the standard cells is generated by using the SOAP [14] algorithm which can handle standard cell placements in rectilinear regions. The SOAP algorithm uses the self organization principle to generate the placement for the standard cells. SOAP is also capable of handling pre-placed cells. We have made a minor modification to the row generation part of the algorithm, which generates rows of standard cells, such that all overlaps between pre-placed cells is eliminated in the final placement of standard cells.

CHAPTER VI

EXPERIMENTAL RESULTS

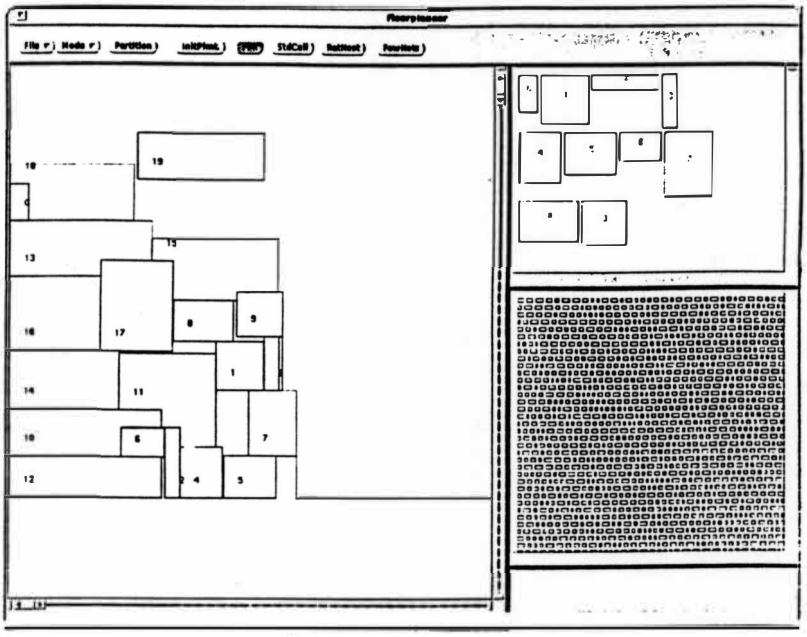

Figure 18. The ARCHITECT Floorplanning Tool With FBR in Action.

The ARCHITECT floorplanning system has been implemented in C, using Xview on a SPARC station 1+. Due to lack of industrial benchmarks, it was tested on randomly generated examples. Figure 18, shows the ARCHITECT floorplanning tool with the FBR algorithm in action.

Table 1, shows the layout areas of the various examples that were tested along with the number of fixed blocks, standard cells and the number of nets. The FBR algorithm drastically reduces white space in the floorplan. The reduction in white space depends on the parameters α and β set by the user. It also depends on the relative number of fixed blocks and standard cells in the layout. The layout areas indicated in Table 1, are for $\alpha = 5$ and $\beta = 30$ units. Usually, the FBR algorithm requires 3 to 4 iterations to converge to the best layout possible with the given values of α and β. Notice that for FP4, the layout has minimum amount

Table 1

Floorplan Areas Generated by ARCHITECT

No.	blks	cells	nets	A_c†	A_f‡	% WS*
F1	3	800	400	40664	44204	8.01
F2	5	900	600	51242	55194	7.17
F3	8	1400	890	75947	83458	9.0
F4	12	5000	1400	221941	234858	5.5
F5	15	1200	750	104153	118356	12.0
F6	10	2200	750	101032	108636	7.0

†Area of blocks & cells, ‡Floorplan Area, *White space

of white space. This is due to the large number of standard cells available in the layout. As the number of blocks increases and the number of standard cells is reduced (FP5), the white space in the floorplan increases. We have noticed that initial size and shape of the blocks have influence on the initial placement but the final floorplan is independent of the initial block sizes.

Figure 19 shows the initial placement for example FP2 that was used as input for the FBR algorithm. Figure 20 shows the floorplan generated by FBR for example FP2. The values of α and β were set at 6 and 15 respectively for all the flexible blocks. Figure 21 shows the final layout generated for the same example with $\alpha = 8$ and $\beta = 8$ units for all blocks. Due to this change in values of α and β, blocks 12, 17 and 20 have been affected. In Figure 20, block 17 can have only 6 edges and hence the vacant space beneath this block cannot be utilized. Block 12 cannot utilize the vacant space underneath it due to the β constraint. Finally, in the IRBBAST problem when the range of block 20 was changed to take advantage of the vacant space over block 13, again the β constraint was violated. Hence, the range of block 20 was not changed. When the values of α was increased to 8 and

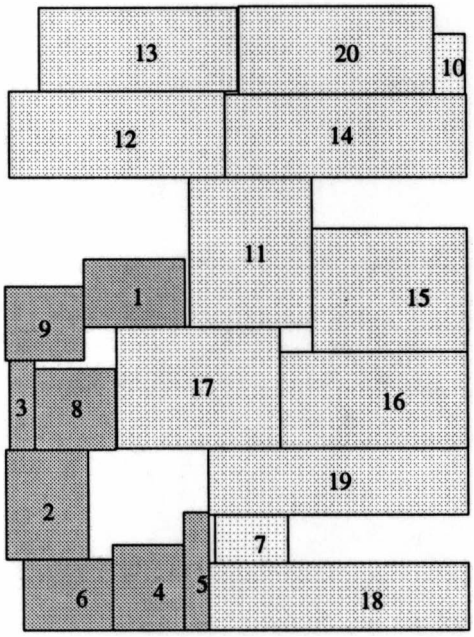

Figure 19. The Initial Placement of FP2 Used as Input for FBR.

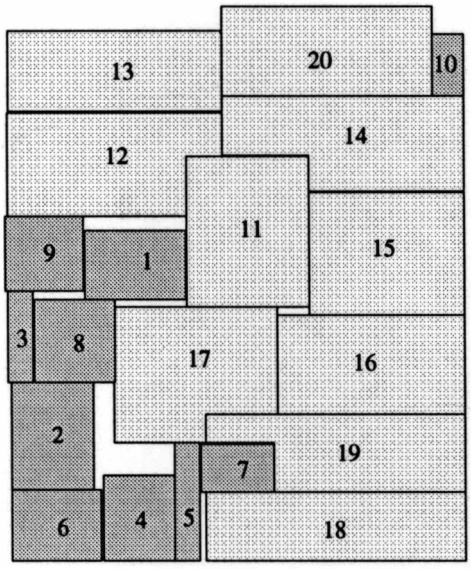

Figure 20. Layout of FP2 for α=6 and β = 15.

range of block 20 was not changed. When the values of α was increased to 8 and the value of β was reduced from 15 to 8, as shown in Figure 21, the vacant space under block 17 could be utilized. The reduction of the value of β allowed block 12 to utilize the vacant space underneath it. Also, the reduction in the value of β, allowed block 20 to change its range in the IRBBAST problem. Hence, when the value of α is increased and the value of β is decreased, the amount of vacant space in the floorplan is reduced and the layout area improves. The FBR algorithm took 185 seconds for generating the floorplan for example FP2. Both Figures 20 and 21 were dumped into xfig format by ARCHITECT and then converted into tex format so that they could easily be incorporated as figures for this thesis.

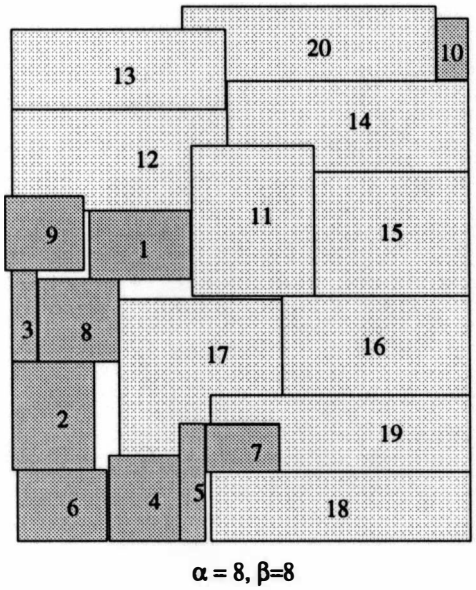

Figure 21. Layout of FP2 for α=8 and β = 8.

Figure 22, shows the decrease in white space area during each iteration of the FBR algorithm for example FP2. Each curve in the figure is for a given value of α and β. It is clear that as the α and β constraints on the blocks are relaxed, the amount of white space in the final layout decreases.

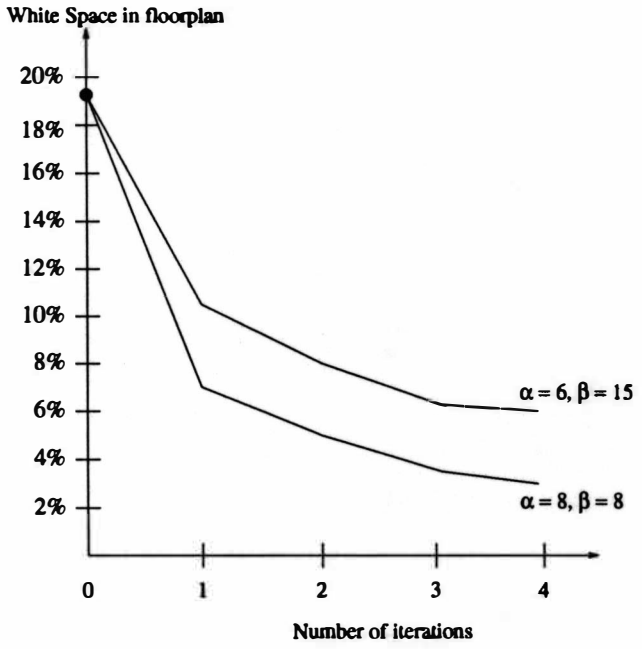

Figure 22. Reduction of White Space During Each Iteration of FBR.

CHAPTER VII

CONCLUSIONS

In recent years, Mixed Block and Cell designs are increasingly becoming popular due to the advantages offered by this design style. The floorplanning algorithms existing for MBC designs do not utilize the flexibility of the standard cell regions while generating the floorplans for MBC designs. In this thesis, we have developed a new floorplanning algorithm for MBC designs. Our floorplanner generates rectilinear shaped flexible blocks which improves the layout area of a design. To the best of our knowledge, this is the first time shapes much more complex then 'L' have been considered. We have defined two new shape parameters, α and β which are used to control the shapes of rectilinear flexible blocks. We have incorporated our algorithm into a tool for generating floorplans for MBC designs. From our experimentation we can conclude the following:

1. The larger the number of standard cells in a design, the lower is the white space in the layout. This is mainly due to the flexibility of the standard cell regions.

2. As we relax the α and β constraints on the blocks in a layout, the white space in the layout tends to decrease.

Our approach can be easily extended to handle fixed blocks with rectilinear shapes.

REFERENCES

[1] G. Vijayan and R. Tsay, "A New Method for Floorplanning Using Topological Constraint Reduction", *IEEE Transactions on Computer-Aided Design,* December, 1991, pp 1494-1501.

[2] S. Sutanthavibul, E. Shragowitz and J. Rosen, "An analytical Approach to Floorplan Design and Optimization", *IEEE Transactions on Computer-Aided Design,* June, 1991, pp 761-769.

[3] S. Dong, J. Cong and C. Liu, "Constrained Floorplan Design for Flexible Blocks", *Proc. of the International Conference on Computer Aided Design,,* 1989, pp 488-491.

[4] K. Kozminski and E. Kinnen, "Rectangular Duals of Planar Graphs", *Networks*, Vol.15, 1985, pp. 145-157.

[5] J. Bhasker and S. Sahni, "A Linear Time Algorithm to Check for the Existence of a Rectangular Dual of a Planar Triangulated Graph", *Networks*, Vol. 17, 1987, pp. 307-317.

[6] W. Dai, B. Eschermann, E. Kuh, and M. Pedram, "Hierarchical placement and floorplanning in BEAR", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8, December 1989, pp. 1335-1349.

[7] T. Wang and D. Wong, "An Optimal Algorithm for Floorplan Area Optimization", *Proc. of the 27th ACM/IEEE Design Automation Conference*, 1990, pp. 180-186.

[8] J. Apte, and, G. Kedem, "Heuristic algorithms for combined standard cell and macro block layouts", *Proc. of the Sixth MIT Conf. on Advanced Research in VLSI*, 1990, pp. 367-385.

[9] K. Ueda, H. Kitazawa, and, I. Harada, "CHAMP: Chip Floor Plan for Hierarchical VLSI Layout Design", *Proc. of IEEE Trans. on Computer-Aided Design*, Vol. CAD-4. No. 1, Jan 1985, pp. 12-22.

[10] M. Upton, K. Samii, and S. Sugiyama, "Integrated Placement for Mixed Macro Cell and Standard Cell Designs", *Proc. of 27th ACM/IEEE Design Automation Conference*, 1990, pp. 32-35.

[11] R. Putatunda, D. Smith, M. Stebnisky, C. Puschak, and P. Patent, "VITAL: Fully Automatic Placement Strategies for Very Large Semicustom Designs", *International Conference on Computer Design*, 1988, pp. 434-439.

[12] C. Sechen, "Chip-Planning, Placement, and Global Routing of Macro/Custom Cell Integrated C ircuits Using Simulated Annealing". *Proc. 25th Design Automation Conference*, 1988, pp. 73-80.

[13] T. Lee, "A Bounded 2D Contour Searching Algorithm for Floorplan Design With Arbitrarily Shaped Rectilinear and Soft Modules", *Proc. of the 30th ACM/IEEE Design Automation Conference*, 1993, pp. 525-530.

[14] Sung-Soo Kim and Chong-Min Kyung, "Circuit Placement on Arbitrarily Shaped Regions Using Self-Organization Principle", *IEEE Transactions on Computer-Aided Design*, Vol.11, No.7, July 1992, pp. 844-854.

[15] M. Garey and D. Johnson, "The Rectilinear Steiner Tree Problem is NP-Complete", *SIAM Journal of Applied Mathematics*, 32, 1977, pp. 826-834.