View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by ScholarBank@NUS

B*-TREE REPRESENTATION BASED THERMAL AND VARIABILITY AWARE FLOORPLANNING FRAMEWORK

SHEFALI SRIVASTAVA
(B.Tech. (Hons), CSJM University, India)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2007

Acknowledgements

First of all, I would like to express my deepest gratitude and appreciation to my supervisor **Dr. Ha Yajun**, without whom, this thesis would not have materialized. I sincerely thank him for his valuable guidance, suggestions and constant support throughout my M.Eng thesis research work. I am grateful that he provided me with the opportunity to work on the cutting-edge technology and the most important research areas in the VLSI domain, which has further enhanced my career prospects in future. His emphasis on reasoning out everything, on clarity in presentation of ideas and on looking at the holistic picture of a problem has always guided me in the right direction. His persistence in conquering difficulties in research and the enthusiasm for work have served as excellent examples for me to follow throughout my career.

Finally, I would also like to thank all those who have directly or indirectly provided advice and assistance during the course of my research work in the National University of Singapore.

Table of Contents

Summary

The evolution of deep submicron technologies has placed a high importance on power dissipation and temperature of the chip. In addition, the increasing design complexity is causing higher levels of uncertainty in design prototyping in the early chip planning stages, thus leading to parameter variations which are posing an ever-increasing challenge to performance analysis of high-speed designs. The purpose of this thesis is to develop an interconnect power and thermal aware, and a variability-aware floorplanner based on B*-Tree representation, to combat the effects of scaling technologies on temperature and variations due to design uncertainty, separately. The thesis consists of two parts which explain our work done in the areas of interconnect power and thermal aware floorplanning, and variability-aware floorplanning as described below.

Interconnect power dissipation is becoming a performance bottleneck in sub micron technologies leading to dramatic rise in chip temperatures which have negative impact on chip performance and reliability. However, most prior work fail to consider the switching activity of interconnects in deriving interconnect power dissipation and in exploring a thermal-aware floorplan. This can result in peak temperatures being underestimated by as much as 15^{o}C according to our experiments. In this work, we present an interconnect power and thermal aware floorplanner that aims at reducing hotspots and distributing temperature evenly across a chip, while optimizing the traditional design metrics, chip area and wirelength. Results demonstrate that our floorplanner is effective in lowering peak temperatures by as much as 20% while providing floorplans that are as compact as the traditional area oriented techniques with just a slight overhead of total wirelength by 2% when testing on five MCNC benchmarks.

The ever increasing growth of design complexity with the scaling of technologies towards the nanometer regime brings with it a challenging increase in the amount of variability due to uncertainty in initial estimates in early phases of the chip planning. With the introduction of variations due to uncertainty in block characteristics such as width, height and aspect ratio, a traditional deterministic floorplanner is unable to take block's variations into account and a variability-aware floorplanner is needed. In this work, we use an affine arithmetic (AA) model to develop a fast and optimized variability-aware floorplanner. The AA model enables a fast and accurate estimation of the variable range of floorplan metrics such as area and wirelength in the presence of variations of each block's dimensions. Compared with the Monte Carlo simulation results, the average errors of mean and range values computed by the proposed method are –0.78% & –12.96% respectively for area, –2.43% & –13.23% respectively for wirelength and up to 100X speed up by testing on five MCNC benchmarks. Our solution to this problem is also interesting to related problems such as warehouse floorplanning.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Aggressive scaling of process technologies towards the nanometer regime has enabled feature sizes to shrink continuously. This allows designers to pack more functionality onto a single die. However, the increased level of integration within a single die imposes rigid constraints on the power consumption budget and hence the temperature profile of the chip. Also, it brings with it a challenging increase in the amount of variability due to uncertainty in initial estimates in the early phases of VLSI design i.e. design prototyping.

1.1.1 Floorplanning

Floorplanning is an important step in the VLSI design process to plan the positions and orientations of a set of circuit modules on a chip in such a way that no blocks overlap and the circuit performance is optimized. It can have drastic impacts on the quality and flexibility of a design such as layout area, wirelength congestion, power density and temperature of the chip. As technology moves into the deep-submicron era, circuit sizes and complexities are growing rapidly and floorplanning has become more important than ever before. With the introduction of uncertainty in the block dimensions at the time of floorplanning, a variability-aware floorplanner can predict the ranges of area and wirelength of a design that impact acceptability assessment of the chip architecture in the early design decision stage.

There are two aspects in general when dealing with the floorplanning problem. The first one is to find an appropriate topological representation (CBL, B*-Tree, O-Tree, TBS, BSG, Sequence Pair, TCG etc.) in the form of a data structure to represent the geometrical relationship among the blocks. The second aspect considers the application of a stochastic search method on the representation to find an optimized floorplan. Most floorplanning algorithms use simulated annealing to search for an optimal solution. Floorplanning has been proven to be a NP-hard problem, hence, it is important to chose a good representation and a searching methodology to perturb the infinite solution space to search for a near optimal floorplan solution in less time. Most of the research is focused on these two aspects of the floorplanning problem.

1.1.2 Thermal and Power Dissipation effects

Interconnects have become the center of attraction in terms of power consumption and performance as the process technology scales into the deep sub micron region. However, interconnects, unlike transistors, have not scaled down exponentially as we move to nano meter era. This has led to an increase in the total capacitance of interconnects and hence dynamic power dissipation despite the introduction of low dielectric materials. Secondly, long interconnects, compared to the scaled transistors, are becoming exceptionally long. In order to keep the delays of these long wires tractable, repeaters and flip-flops are inserted to prevent performance degradation. However, these additional components have detrimental impacts on interconnect power dissipation.

Power density directly translates into heat which may lead to a significant increase in chip temperature. As a result, the temperature in modern high performance VLSI circuits increases dramatically due to smaller feature size, higher packing density and rising power consumption. The hotspot in a modern chip might have a temperature of more than 100^{o}C while the intrachip temperature differentials can be larger than 10~20^{o}C [41]. Temperature can have a dramatic impact on circuit performance, power, and reliability: MOS current drive capability decreases approximately 4% for every 10^{o}C temperature increase while the leakage current increases exponentially with the temperature increase resulting in thermal runaways. The interconnect resistance also becomes larger with increasing temperature. For example, the resistivity of copper increases by 39% from 20^{o}C to 120^{o}C. Higher resistivity causes longer interconnect RC delay and hence performance degradation. The interconnect (Elmore) delay increases approximately 5% for every 10^{o}C increase in temperature [1]. Higher temperatures accelerate electro migration failures and reduce the lifetime of the device. Finally, high temperature of the chip makes cooling solutions significantly more expensive. Therefore, it is very important to eliminate hotspots and have a thermal balanced design. Power-aware design alone is not able to address the temperature challenge because the thermal distribution profile depends on not only the power density but also the physical size and location of each functional block. Therefore, even though it is related to the power-aware design area, thermal-aware design itself is a distinct and important research area.

1.1.3 Design Prototyping

Design prototyping has gathered much attention recently due to increasing complexity of VLSI designs and the need for area and performance measures early in the chip planning stage. Based on the assessment of the above measures, the chip architecture is revisited and it is partially or completely redesigned accordingly to meet the design specifications. This saves a lot of time and cost incurred otherwise due to changes made after the actual implementation. Since design prototyping occurs very early in the design phase, uncertainty exists in the circuit physical dimensions and the technology library cells. This poses a challenging task to the designer to have initial estimates about area and performance parameters of the chip. Therefore, we need to correctly model these uncertainty variations not only to determine the correct expected circuit area and performance of a design but also to correctly optimize the design such that the percentage of parts that meet a specified performance target is maximized.

1.2 Problem Definition

The work in the thesis is inspired from the concerns at rising trends in accurate thermal-conscious mechanisms and the impact of variations due to design uncertainty in early planning stage of a chip fabricated with sub-micron technologies. Designers are looking at developing new methods to tackle these problems at an early design stage so that unnecessary work may be avoided at later stages. Floorplanning has been a major focus of attention and research since it can impact many important design decisions at an early stage.

Many thermal-aware floorplanners exist that estimate the temperature of the chip and help to reduce hotspots by clever floorplanning techniques. However, with the onset of high switching activity circuits i.e. circuits which have high usage of specific interconnects and increasing interconnect power dissipation, previous floorplanners fail to provide accurate temperature estimates. Therefore, we need to consider the switching activity of different interconnects in deriving the total power dissipation in a circuit. Since the temperature of the chip depends largely on the power distribution profile in a chip, neglecting any of the factors responsible for power dissipation will give a pessimistic analysis of temperature estimates.

As designs are getting more complex, it is difficult to have the entire design (the netlist and the database library) completely specified and available at the time of floorplanning. This introduces uncertainty in block dimensions which then need to be specified by a range of values for further evaluation. Floorplanning with uncertainty is the process of obtaining an accurate floorplan with missing data. With the introduction of variations in block characteristics such as width, height and aspect ratio, due to design uncertainty, a traditional deterministic floorplanner is unable to take block variations into account and a variability-aware floorplanner is needed. The traditional floorplanners optimize design metrics to find one best *fixed* floorplan for blocks with only *fixed* block characteristics. For example, an area driven traditional floorplanner might perform optimizations to find one fixed floorplan that yields the minimum total area. It assumes that each block has fixed block area or aspect ratio when estimating the total chip area. However, the variability-aware floorplanner optimizes design metrics to find one best *relative* floorplan for all blocks with *variable* block characteristics. For example, an area driven variability-aware floorplanner might perform optimizations to find one relative floorplan that yields the minimum average total area for all possible variations in the range. It assumes that each block has variable block area or aspect ratio when estimating the total chip area. As a result, with the introduction of variability in the dimensions for each module, the conventional approach fails to estimate the ranges of area, wirelength and other floorplan metrics.

To estimate the range caused by variations, two types of brute-force approaches can be tried. The first approach assumes the worst-case magnitude for each variation but the resulting floorplan will be very pessimistic. The second approach performs Monte Carlo simulations to consider enough points in a very large variation space, although the results are relatively accurate but the simulation time is extremely long. Both approaches are not ideal to have a fast and accurate estimation of ranges.

To provide a fast yet accurate estimation of ranges caused by variations, the related area in statistical static timing analysis (STA) uses analytical approaches to find closed-form expressions for the distributions of circuit delays. These methods use normal distributions, interval valued analysis, probabilistic intervals or mathematical statistical models. We believe a similar approach is needed to quickly and accurately estimate the floorplan metric ranges.

Thus, our problem definition consists of developing a floorplanning algorithm with the objective of minimizing chip area and wirelength together with achieving two major goals as outlined below:

- Accurate chip thermal modeling with the consideration of switching activity of interconnects and netlength in determining interconnect power dissipation together with the block power dissipation in a chip. The goal of floorplanning algorithm is to evenly spread out the temperature on a chip and thus reduce hotspots.
- Determining the range of total area and wirelength under the presence of variations in dimensions of modules. The goal is to determine the best relative floorplan with the smallest range and average for all possible variations in total area and wirelength.

1.3 Our Contributions

We develop a floorplanner based on B*-Tree representation and use the simulated annealing algorithm to get the best floorplan layout meeting the criteria as outlined in the problem definition given earlier. The major contributions of the thesis are two fold.

- First, our work incorporates the feature of switching activity of interconnects while calculating the power densities across the entire chip and shows that it can lead to different power density profiles depending on values of switching activity, keeping other parameters like positions and dimensions of functional blocks and interconnect lengths constant. This results in significant change in chip temperature leading to localized hot spots. We then develop an interconnect power and thermal aware floorplanner that aims at reducing hotspots and distributing temperature evenly across a chip while optimizing the traditional design metrics, chip area and wirelength. A thermal modeling tool called HotSpot is used to determine the chip temperature profile. Our work is novel with regard to the different floorplanning algorithms developed till recently for chip temperature estimates as none of the previous works take switching activity of the interconnects into account for determining the temperature of the chip. Results in our work indicate that excluding switching activity in interconnect power determination can result in peak temperatures to be under estimated by as much as 15^{o}C. Our floorplanner is effective in lowering peak temperatures by as much as 20% while providing floorplans that are as compact as the traditional area oriented techniques with a slight overhead of total wirelength by 2% when testing on five MCNC benchmarks.

- Second, we present a mathematical model using affine arithmetic to estimate the range of floorplan metrics such as chip area and wirelength under the given variations in dimensions for each module. Our work is a breakthrough with reference to earlier related work that considers variations of width and height of each block to be limited to certain discrete values in a range. Also, we use more advanced analytical technique i.e. affine arithmetic to perform range calculations on a set of MCNC benchmark circuits. Moreover, we also determine range of wirelength in addition to the range of area. We determine the most feasible B*-Tree as the solution which can give the lowest range of values lying as close as possible to the minimum average values of area and wirelength for a given set of variations. We then run Monte Carlo simulations to verify the accuracy of our affine arithmetic model. Experimental results show that, compared with the Monte Carlo simulation results, the average errors of mean and range values computed by the proposed method are –0.78% & –12.96% respectively for area, –2.43% & –13.23% respectively for wirelength and up to 100X speed up by testing on five MCNC benchmarks. Our solution to this problem is also interesting to other related problems such as warehouse floorplanning.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 gives the background information on floorplanning, thermal and power dissipation effects, and variations due to design uncertainty, which provide the base and framework for the work done in this thesis. The chapter gives an overview of floorplanning concepts including the description of B*-Tree and simulated annealing algorithm which have been used in our work. Further, it describes the sources of power dissipation in a chip together with the role of interconnects in determining power dissipation, and hence the temperature estimation of the chip using a thermal modeling tool called Hotspot. Finally, the chapter gives a brief overview of Monte Carlo simulation method, numerical computation methods i.e. interval arithmetic and affine arithmetic which have been used for modeling variations. Chapter 3 reviews the previous work which has served as a source of motivation for this research work and formally describes the problem definition, solution approach and experimental results for our contribution towards making thermal-aware floorplanning more accurate with the incorporation of switching activity of interconnects in deriving the power dissipation in a chip. Chapter 4 presents the formal problem formulation for our contribution towards developing a variability-aware floorplanner based on affine arithmetic to enable a fast and accurate estimation of the variable range of floorplan metrics such as area and wirelength in the presence of variations of block dimensions due to design uncertainty. The chapter reviews the previous related work, explains our variability-aware floorplanning algorithm in detail and presents the experimental results based on both the Monte Carlo simulation approach and our approach. Chapter 5 draws the conclusions and provides future research directions.

Chapter 2

Background and Related Work

In this chapter, we discuss some preliminary topics which will provide the necessary ground framework for the work done in this thesis. The chapter is organized as follows. Section 2.1 briefly discusses floorplanning concepts and the typical approach followed to solve a floorplanning problem. It gives an overview of various topological representations, block and net model including pin assignment and wirelength estimation method, and various floorplanning algorithms. Section 2.2 discusses various causes of power dissipation including interconnect power dissipation. It also explains the method to estimate temperature together with a brief description of thermal modeling tool called Hotspot. Section 2.3 explains Monte Carlo simulation method, and mathematical models like interval arithmetic and affine arithmetic, which are used for modeling variations.

2.1 Floorplanning

Floorplanning is an important step in physical design of VLSI circuits to plan the positions and orientations of a set of circuit modules on a chip in order to optimize the circuit performance. The quality of the floorplan solution depends largely on the choice of topological representation and the floorplanning algorithm selected to search over an infinite solution space.

2.1.1 Topological Representation

In order to floorplan a circuit design, an abstract representation is needed to represent the geometrical relation or topologies among blocks so that some algorithms can be applied on to solve the problem. This abstract representation is called topological representation and is commonly specified by a rectangular dissection of the floorplan region. Floorplans can be divided into two categories, the *slicing floorplans* [4], [5] and the *non-slicing floorplans* [6], [7], [8] and [9].

Slicing floorplan is a rectangular dissection that can be obtained by recursively cutting a rectangle horizontally or vertically into two smaller rectangles. Slicing floorplans are represented by slicing structures which can be modeled by a binary tree with *n* leaves and *n-i* nodes where each node represents a vertical cut line or a horizontal cut line and each leaf a basic rectangle. Figure 2.1 shows the slicing floorplan and its corresponding binary tree.

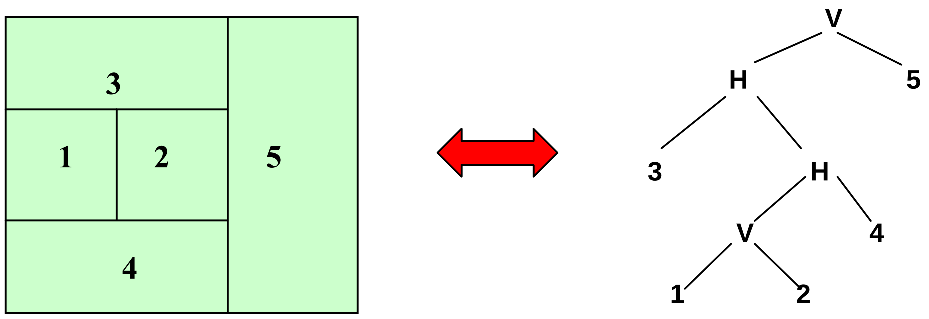


Figure 2.1: Slicing floorplan and its corresponding binary tree.

Non-slicing floorplans are further categorized into *mosaic floorplans* and *general floorplans*. *Mosaic floorplan* is one which is dissected into exactly *n* rooms so that each room is occupied by one and only one block. E.g. a wheel structure as shown in Figure 2.2. In addition, there is no crossing cut in the mosaic floorplan.

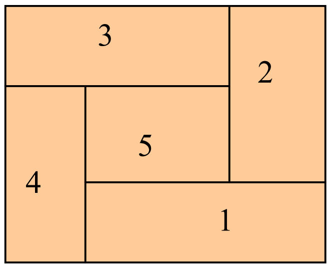


Figure 2.2: Mosaic floorplan (wheel structure).

General floorplan is similar to mosaic floorplan in that non-slicing structures are allowed. However, the floorplan region can be dissected into more than *n* rooms such that some rooms are empty, i.e. not occupied by any block as in Figure 2.3.

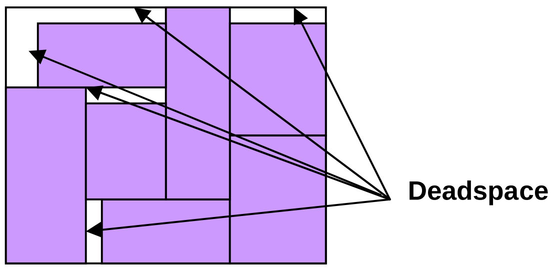


Figure 2.3: General floorplan.

Deadspace of a floorplan is the space that is wasted as shown in Figure 2.3. Minimizing area is the same as minimizing deadspace. Deadspace percentage is computed as

$$\frac{(A - \sum A_i)}{\sum A_i} \times 100\% \qquad (2.1)$$

where A_i is the area of each block i and A is the total area of the floorplan.

Slicing floorplan is a special case of mosaic floorplan and mosaic floorplan is a special case of general floorplan. The relationship among the solution spaces of slicing, mosaic and general floorplans is illustrated in Figure 2.4 on next page.

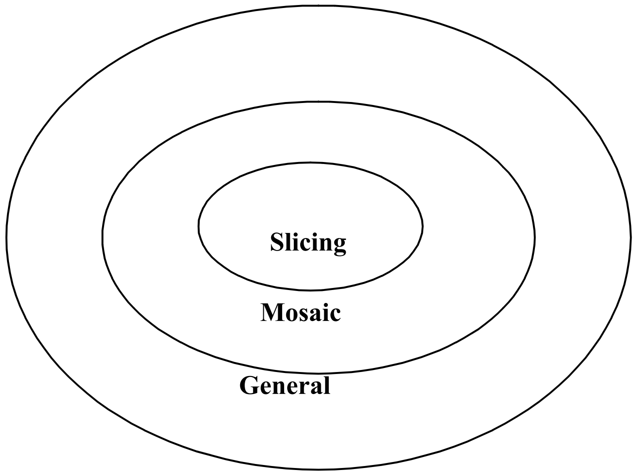


Figure 2.4: Relationship among the solution spaces of slicing, mosaic and general floorplans.

Various topological representations like Normalized Polish Expression, B*-Tree [10], O-Tree [6], Sequence Pair [11], Corner Block List (CBL) [12], Bounded Sliceline Grid (BSG) [8], Transitive Closure Graph (TCG) [13] etc. have been proposed to represent slicing and non-slicing floorplans. We briefly describe some of the most popular representations below.

2.1.1.1 Normalized Polish Expression (NPE)

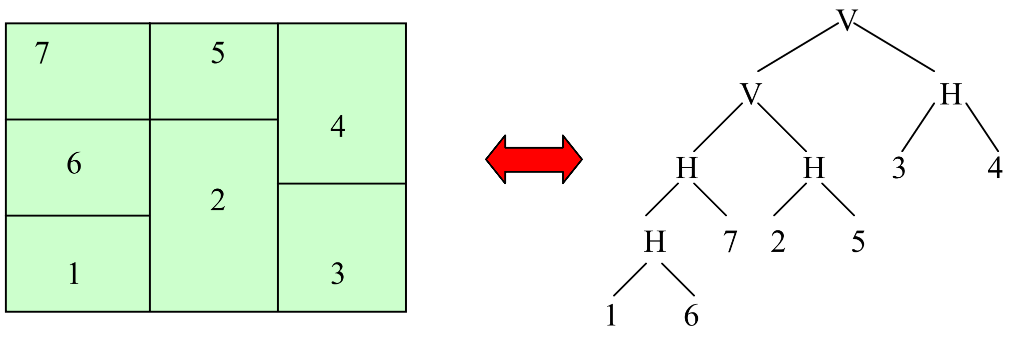
Normalized polish expression [5] proposed by Wong and Liu is used for representing slicing floorplans. NPE removes the redundancy in the binary tree representation, which is due to the existence of more than one binary tree corresponding to the same slicing floorplan. An ideal data representation is one which is able to represent all possible combinations of floorplan without having two or more data representations that correspond to the same floorplan.

An expression, $E = e_1, e_2, \ldots, e_{2n-1},$ where $e_i \in \{1, 2, \ldots, n, H, V\}$, $1 \leq i \leq 2n\text{-}1$, is a polish expression of length $2n$-1 iff

- Every operand j, $1 \leq j \leq n$, appears exactly once in the expression, and
- The expression E has the balloting property, i.e. for every sub-expression $E_i = e_1, e_2, \ldots, e_i$, $1 \leq i \leq 2n\text{-}1$, the number of operands is greater than the number of operators.

A polish expression is said to be a normalized polish expression iff E has no consecutive H's and V's (e.g. 16H7H25HV34HV).

We can view a normalized polish expression as a bottom UP description of a slicing structure. In fact, we can interpret the symbols H and V as two binary operators between slicing structures. If A and B are slicing structures, we can interpret AHB and AVB as the resulting slicing structures obtained by placing B on top of A, and B to the right of A, respectively. A postorder traversal of the slicing tree results in a NPE with V and H as the operators, and the basic rectangles as operands (See Figure 2.5). This expression specifies how to build the final slicing structure from smaller ones. Figure 2.5 shows the slicing floorplan, its NPE and the corresponding slicing tree.



NPE = 16H7H25HV34HV

Figure 2.5: Slicing floorplan, its NPE and the corresponding slicing tree.

2.1.1.2 Sequence Pair

Murata et al. [11] proposed the sequence-pair representation for rectangular module placement. The main idea is to use a sequence pair to represent the geometric relation of modules, place the modules on a grid structure and construct corresponding constraint graphs to evaluate cost. This representation requires $2n[log\ n]$ space to encode a sequence pair and there are $(n!)^2$ combinations in total where n is the number of modules. Further, the transformation between a sequence pair and a placement takes $O(nlogn)$ time.

A sequence pair (Γ+, Γ-) is a pair of sequences of elements representing a list of blocks. The two sequences Γ+ and Γ- are permutations of a given block set. The sequence pair structure is actually a meta grid. Given a sequence pair, one can construct a 45 degree oblique grid as shown in Figure 2.6 (a). For every block, the plane is divided by the two crossing slope lines into four cones as shown in Figure 2.6 (b). Block 2 is in the right cone of block 1, then it is to the right of block 1 [see Figure 2.6 (c)]. In general, the relative positions between any two blocks *a* and *b* can be derived from a sequence pair (Γ+, Γ-) by the following rules.

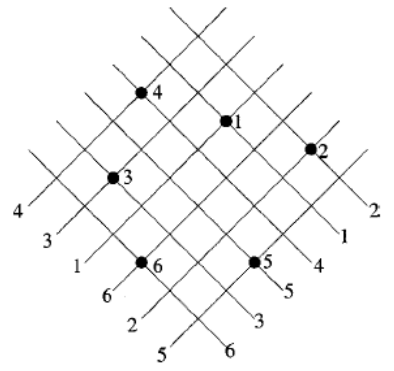
- Horizontal constraint:

  If (Γ+, Γ-) = (<…*a*, …, *b*…>, <…*a*, …, *b*…>), block *b* is at the right side of block *a*.

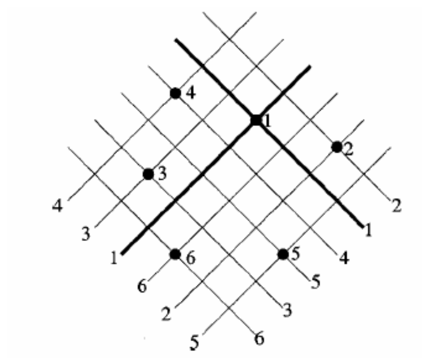
- Vertical constraint:

  If (Γ+, Γ-) = (<…*a*, …, *b*…>, <…*b*, …, *a*…>), block *b* is below block *a*.

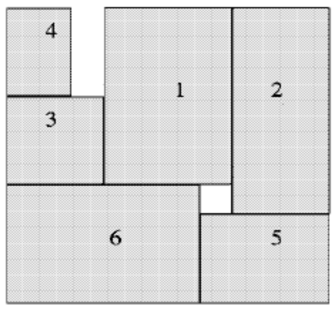
For example, (Γ+, Γ-) = (<431625>, <635412>) is a sequence pair of block set {1, 2, 3, 4, 5, 6}. Figure 2.6 shows the oblique grid and packing of the sequence pair (<431625>, <635412>).



(a)



(b)

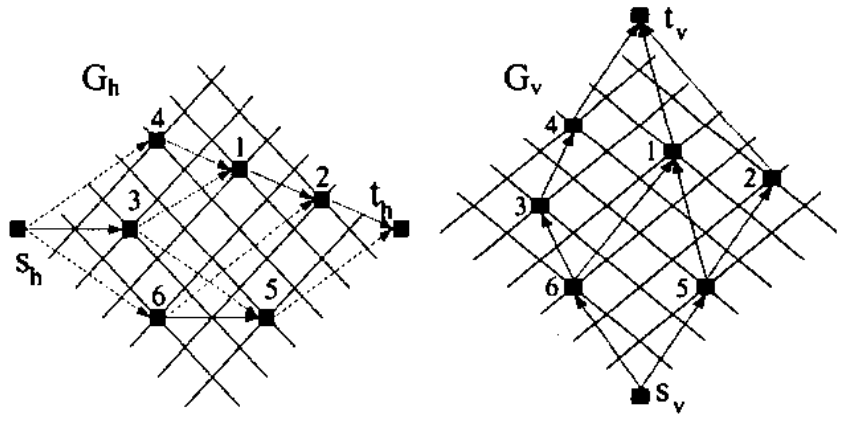


(c)

Figure 2.6: (a) Oblique grid for sequence pair (<4 3 1 6 2 5>, <6 3 5 4 1 2>) (b) Four cones of block 1 (c) Corresponding packing. Dimensions for the six blocks are: 1 (4 × 6), 2 (3 × 7), 3 (3 × 3), 4 (2 × 3), 5 (4 × 3) and 6 (6 × 4).

Evaluation of a Sequence Pair

In order to evaluate the corresponding floorplan of a sequence pair and determine the position of each block, Murata et al. [11] used two weighted directed constraint graphs G_h and G_v that are constructed according to horizontal and vertical constraints of a sequence pair. Then, the longest path algorithm is invoked to determine the longest paths of the two graphs. The longest paths of G_h and G_v are the width and height of the corresponding placement of the sequence pair respectively. Figures 2.7 (a) and (b) are examples of the weighted directed constraint graphs G_h and G_v of the sequence pair (Γ+, Γ-) = (<431625>, <635412>).



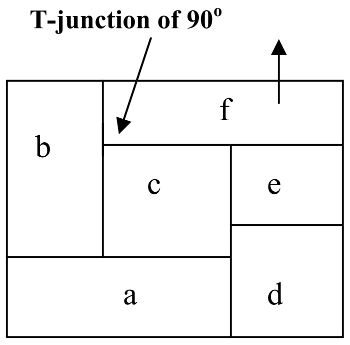
(a) **(b)**

Figure 2.7: The horizontal and vertical constraint graphs of a sequence pair (<431625>, <635412>) (a) Horizontal constraint graph (b) Vertical constraint graph.

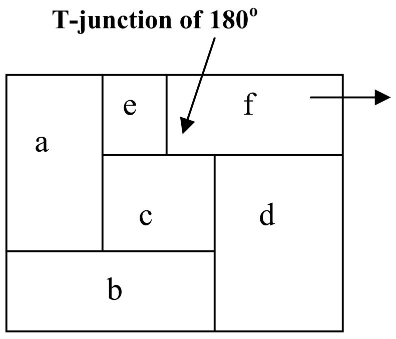
The time complexity of the longest path algorithm is $O(n^2)$ where n is the number of blocks. It had been proved that the longest path of G_h is equivalent to the longest common subsequence of Γ+ and Γ-, and the longest path of G_v is equivalent to the longest common subsequence of $\Gamma+^R$ and Γ- where $\Gamma+^R$ is the reverse sequence of Γ+. Based on the theorm, Tang et al. [7] used an effective data structure to determine the longest common subsequence of a sequence pair in $O(nloglogn)$ time. They use a complete binary tree and a doubly-linked list to determine the longest common subsequence of a sequence pair. The doubly linked list is used to keep the longest common subsequence during the evaluating process and a complete binary tree is used to find the position of a new element which will be inserted into the doubly-linked list.

2.1.1.3 Corner Block List (CBL)

Corner Block List is proposed by Hong et al. [12] to represent mosaic floorplan. The corner block (CB) is the block at upper-right corner of the floorplan. The left and bottom bounding segments of CB form a T-junction.



(a) Vertical CB



(b) Horizontal CB

Figure 2.8: Two different kinds of T-junction and orientation of the corner block (CB) "f".

The orientation of CB is defined by the orientation of its T-junction. The T-junction has only two types of orientations: *T* rotated by 90 degrees (Figure 2.8 (a)) and by 180 degrees (Figure 2.8 (b)) counterclockwise respectively. If *T* is rotated 90 degrees counterclockwise, the CB is vertically oriented, and its corresponding entry in list *L* is set by a "0". Otherwise, the CB is horizontally oriented, and the entry in list *L* is set by a "1". For example, in Figure 2.8 (a), the orientation of corner block "f" is vertical and is denoted by "0" whereas in Figure 2.8 (b), the orientation of "f" is horizontal and is denoted by "1".

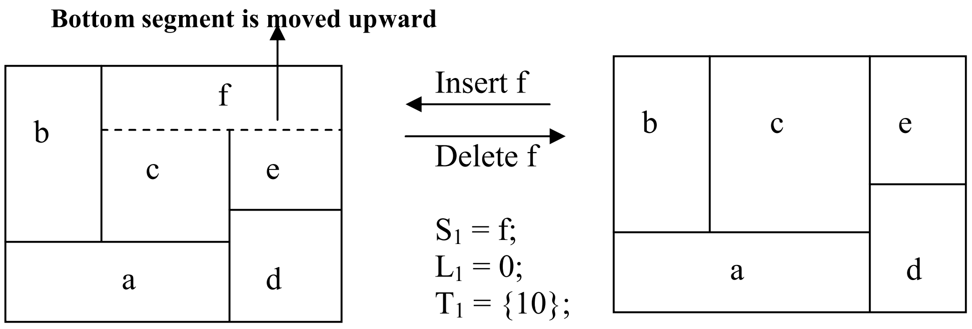


Figure 2.9: Deletion/Insertion of vertically oriented corner block "f".

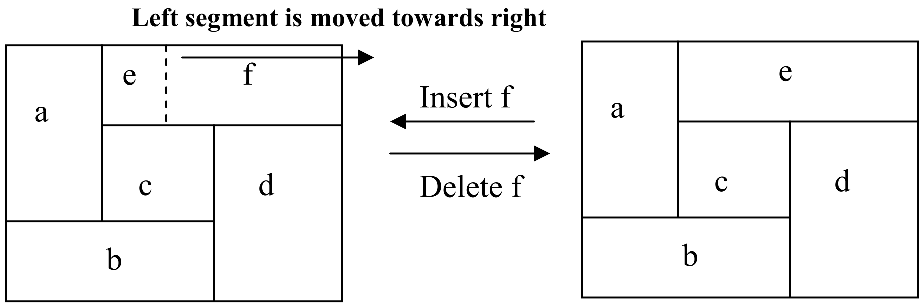


Figure 2.10: Deletion/Insertion of horizontally oriented corner block "f".

Corner Block Deletion

The core idea of CBL representation is embodied in the corner block deletion operation. The way to delete a CB depends on its orientation. To delete a CB which is vertically oriented, its bottom segment is shifted to the top boundary of the floorplan and the attached T-junctions (if any) are pulled along with the segment. Figure 2.9 illustrates this operation. The corner block "f" is vertically oriented, thus, in order to delete this CB, the bottom segment of its room is shifted to the top boundary and the attached T-junction (in this case there is only one attached T-junction) is pulled along with the bottom segment. If the CB is horizontally oriented, the left segment of its room is shifted to the right boundary of the floorplan and the attached T-junctions (if any) are pulled along with the segment. Figure 2.10 illustrates the deletion of horizontally oriented CB "f".

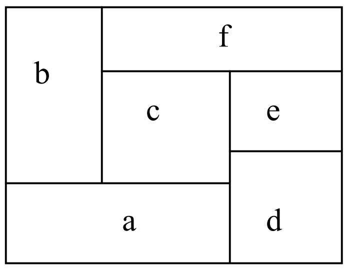
Corner Block Insertion

Corner block deletion is the inverse of deletion. If the inserted CB is vertically oriented, the horizontal segment from the top of the floorplan covering a designated numbers of T-junctions is pushed down in order to create a room for the inserting CB. Figures 2.9 and 2.10 illustrate the insertion operation of corner block “f” starting from the floorplan shown in the right and obtaining the floorplan as shown in the left after insertion of CB “f”. If the corner block is horizontally oriented, the operation is similar to those of vertical oriented but instead of pushing the top segment, the vertical segment at the right of the floorplan is pushed towards left.

It can be observed that the floorplan still remains mosaic after the deletion or insertion operation.

Transformation from floorplan to CBL

CBL list is constructed by recursive deletion of CBs in the floorplan until there is no CB left in the floorplan. For each deletion of a CB, its name is recorded in list *S*, its orientation recorded in list *L* and the number of its attached T-junctions is recorded by the same number of successive “1”s ended by a “ 0” in a binary list T_i (Figure 2.11). At the end of deletions of all CBs, three lists are obtained: the block name list $S = \{M_n, M_{n-1}, ..., M_1\}$, the orientation list $L = \{L_n, L_{n-1}, ..., L_2\}$ and the T-junction list $T = \{T_n, T_{n-1}, ..., T_2\}$. Then each list is reversed and all the items of the T-junction list are combined into a single binary vector *T*. The triple (*S, L, T*) is called a corner block list.



Corner Block List

S = {a, b, c, d, e, f};

L = {0, 1, 1, 0, 0};

T = {0 10 0 0 10};

Figure 2.11: A CBL list and the resultant floorplan.

Transformation from CBL to floorplan

Construction of the floorplan from a CBL is the inverse process. Blocks are inserted in turn either from the right for vertical orientation or from the top for horizontal orientation, covering required number of T-junctions given by the corresponding entry in list *T*. Figure 2.11 illustrates the resultant floorplan of a corner block list.

2.1.1.4 B*-Tree

We shall review the B*-Tree representation in this section. Chang et al. [10] presented a binary tree based representation for a left and bottom compacted placement called B*-Tree and showed its superior properties for operations. Given a placement *P*, we can construct a unique B*-Tree in linear time by using a recursive procedure similar to the depth first search (DFS) algorithm. Each node n_i in a B*-Tree denotes a module. The root of a B*-Tree corresponds to the module on the bottom-left corner. The left child n_j of a node n_i denotes the module m_j that is the lowest adjacent module on the right-hand side of m_i i.e.

$$x_j = x_i + w_i \qquad (2.2)$$

The right child n_k of a node n_i denotes the module m_k that is the lowest visible module above m_i and with the same *x* co-ordinate as m_i i.e.

$$x_k = x_i \qquad (2.3)$$

Figures 2.12 (a) and (b) show a placement and its corresponding B*-Tree respectively. The root n_0 of the B*-Tree in Figure 2.12 (b) denotes that m_0 is the module on the bottom-left corner of the placement. For node n_3 in the B*tree, n_3 has a left child n_4 which means that module m_4 is the lowest adjacent module in the right-hand side of module m_3 (i.e. $x_4 = x_3 + w_3$). n_7 is the right child of n_3 since module m_7 is the visible module over module m_3 and the two modules have the same x co-ordinate ($x_7 = x_3$).

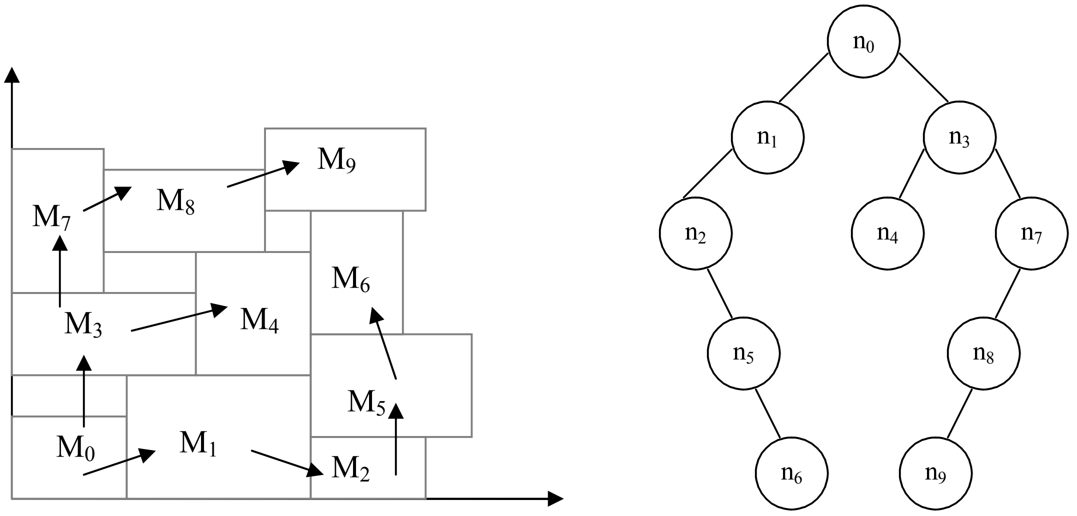


Figure 2.12: (a) A placement (b) The corresponding B*-Tree.

We shall show the procedure to get the placement from a B*-Tree. We first define a permutation π which is the label sequence when we traverse the tree in depth-first search order. The first element in permutation π is the root of tree. We now introduce a contour structure which is used by Guo et al. in [6]. The contour structure is a doubly linked list of modules, which describes the contour line in the current compaction direction. Without the contour structure, the runtime for placing a new module is linear to the number of modules. By maintaining the contour structure, the y co-ordinate for a newly inserted module can be computed in $O(1)$ time. For each module m_i, let $\psi(i)$ be the set of modules m_k with its order lower than m_i in permutation π and interval $(x_k, x_k + w_k)$ overlaps interval $(x_i, x_i + w_i)$ by a non-zero length. If $\psi(i)$ is non-empty, we have

$$y_i = max_{k \varepsilon \psi(i)} y_k + h_k \tag{2.4}$$

Otherwise $$y_i = 0 \tag{2.5}$$

The algorithm for finding the placement from a corresponding B*-Tree is outlined in Figure 2.13 below. It uses a contour structure to reduce the run time for finding the *y* co-ordinate of a module while solving the equations (2.4) and (2.5).

```
Input: B*-Tree(π [0:n])
Output: Placement with position (x_i, y_i) for each module m_i
Begin
    Set perm = 1
    Set contour = NULL
    Set current_contour = 0
    For code = 0 to n−1
        if code = 0 then
            Set current_module = π [perm]
            If current_contour = 0 then
                Set x[current_module] = x[current_contour] + w[current_contour]
            Else set x[curent_module] = 0
            End if
            Set y[current_module] = find_max_y (contour, current_module)
            Update_contour (contour, current_module)
            Set current_contour = current_module
            Set perm = perm + 1
        Else set current_contour = prev [current_contour]
        End if
    End for
End.
```

Figure 2.13: Pseudo code of algorithm for finding the placement from a corresponding B*-Tree.

We use a variable *current_contour* to record the module where we want to insert the next module in the contour. Figure 2.14 shows how *find_max_y* determines the *y* co-ordinate of current module and how *update_contour* updates the contour structure when we add a new module m_8 to the placement. The old contour is composed of modules m_7, m_3, m_4, m_6 and m_5. After m_8 is placed, the new contour becomes m_7, m_8, m_4, m_6 and m_5. Note that we only need to search modules m_3 and m_4 to get m_8's y co-ordinate y_8 with the contour structure.

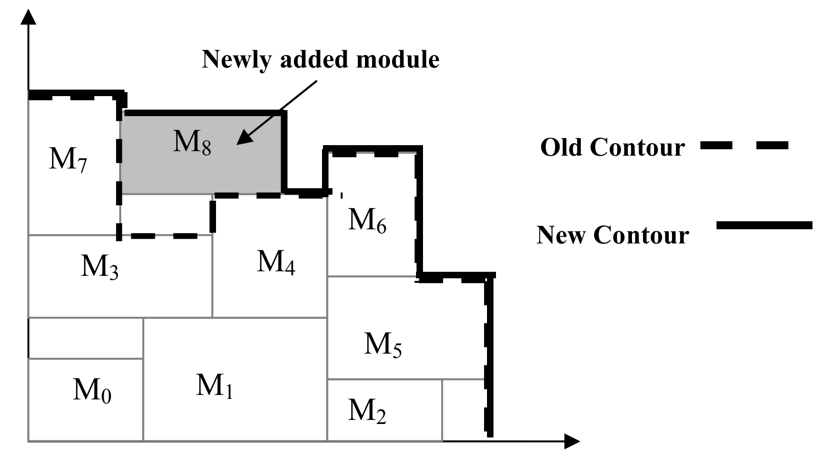


Figure 2.14: Adding a new module on top, we search the contour from left to right and update it with the top boundary of the new module.

We perturb a B*-Tree (a feasible solution) to another B*-Tree by using the following four operations.

1 **Op1**: Rotate a module.

2 **Op2**: Move a module to another place.

3 **Op3**: Swap two modules.

To cope with rotated modules while performing Op1, when inserting a deleted node into a B*-Tree, we can perform the operation twice at each position to find a better solution, one for the original orientation, and the other for the rotated one. Op2 deletes and inserts a module. For the deleted node associated with a rectangular module, we simply delete the node from the B*-Tree. Op2 and Op3 need to apply the Insert and Delete operations for inserting and deleting a node to and from a B*-Tree. We explain the two operations as below.

Deletion

There are three cases for the deletion operation.

1 Case 1: A leaf node.

2 Case 2: A node with one child.

3 Case 3: A node with two children.

In Case 1, we simply delete the target leaf node. In Case 2, we remove the target node and then place its only child at the position of the removed node. The tree update can be performed in $O(1)$ time. In Case 3, we replace the target node n_t by either its right child or left child n_c. Then, we move a child of n_c to the original position of n_c. The process proceeds until the corresponding leaf node is handled. It is obvious that such a deletion operation requires $O(h)$ time where h is the height of the B*-Tree. Note that in Cases 2 and 3, the relative positions of the modules might be changed after the operation, and thus, we might need to reconstruct a corresponding placement for further processing.

Insertion

When adding a module, we may place it around some module. We define two types of positions as follows.

1 *Internal position:* A position between two nodes in a B*-Tree.

2 *External position:* A position pointed by a NULL pointer.

Both these positions can be used for inserting a new node.

We explain the perturbation process with the help of an example in Figure 2.15

that illustrates all the three moves namely Op1, Op2 and Op3. We start with an initial placement and then perform the three operations in the order, i.e. swapping two modules, moving a module from one place to another and rotating a module successively. The corresponding B*-Tree for each placement obtained after the perturbation process is given below it as shown in the Figure 2.15. Note that the rotation operation does not change the configuration of the B*-Tree, only the block orientation changes in the same position in the placement.

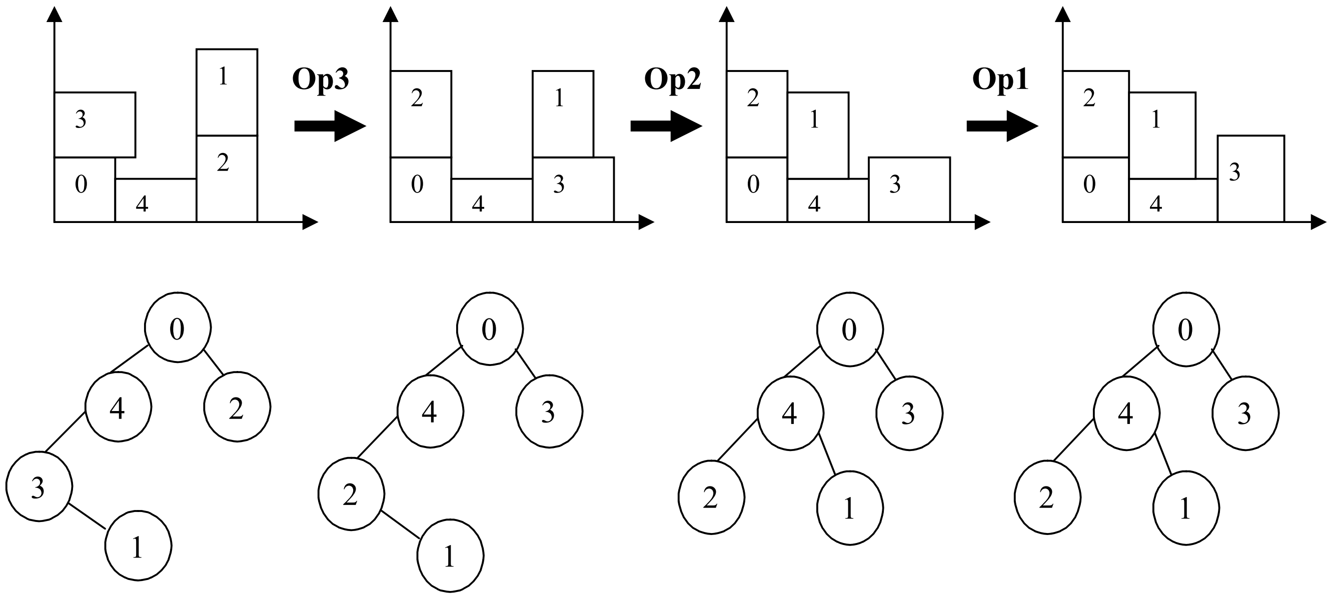


Figure 2.15: Example showing the perturbations process on placement and its corresponding B*-Tree.

We summarize the advantages of B*-Tree as follows:

1. Based on ordered binary trees, B*-Trees are very easy for implementation and can perform the respective primitive tree operations: *search*, *insertion* and *deletion* in only $O(1)$, $O(1)$ and $O(n)$ times while existing representations for non-slicing floorplans need at least $O(n)$ time for each of these operations where n is the number of modules.

2. B*-Tree is very flexible for handling the floorplanning problems with various types of modules (e.g. hard, pre-placed, soft, and rectilinear modules) directly and efficiently. They can handle non-slicing structures.
3. The correspondence between an admissible placement (i.e. which is compacted and can neither move down nor move left) and its induced B*-Tree is 1-to-1 (i.e. no redundancy). Further, the transformation between them takes only linear time.
4. B*-Trees do not need to construct constraint graphs for area cost evaluation. The area cost after exchanging two modules can be recomputed incrementally on a B*-Tree. Specifically, the modules ahead of the exchanged modules in the depth-first search (DFS) of a B*-Tree remain unchanged. Therefore, we need to consider only the modules behind the exchanged ones for cost update.
5. The solution space is smaller, i.e. $O(n!2^{2n-2}/n^{1.5})$.

2.1.1.4 Comparison of different topological representations

The strength of each topological representation can be roughly evaluated by looking at the upper bound of their solution space. Usually, lower the bound, better a representation. Another way of evaluating the performance of the floorplan representation is by looking at its time complexity to transform a floorplan to a placement configuration. A good topological representation should be easily transformed into an actual placement. We summarize the strengths and weaknesses of various topological representations used more popularly in Table 2.1.

Table 2.1. Comparisons among the solution spaces and time complexity of various floorplan representations.

Data Structure	Solution Space	Time complexity
NPE	$O(n!2^{3n}/n^{1.5})$	$O(n)$
B*-Tree	$O(n!2^{2n-2}/n^{1.5})$	$O(n)$
O-Tree	$O(n!2^{2n-2}/n^{1.5})$	$O(n)$
Sequence Pair	$(n!)^2$	$O(n^2)$
Corner Block List (CBL)	$O(n!2^{3m})$	$O(n)$
Transitive Closure Graph (TCG)	$(n!)^2$	$O(n^2)$
Bounded Sliceline Grid (BSG)	$n!C(n^2,n)$	$O(n^2)$

2.1.2 Block and Net Model

Floorplan layout shows the locations of blocks and terminals. Characteristics and arrangement of blocks as well as terminals need to be determined accurately. Net list is described by a net model which also explains how to estimate wirelength.

2.1.2.1 Block Model and Pin Assignment

1 Origin for chip is at (0, 0). Blocks are specified by their lower left (x, y) co-ordinates (positive numbers), and height and width as shown in Figure 2.16.

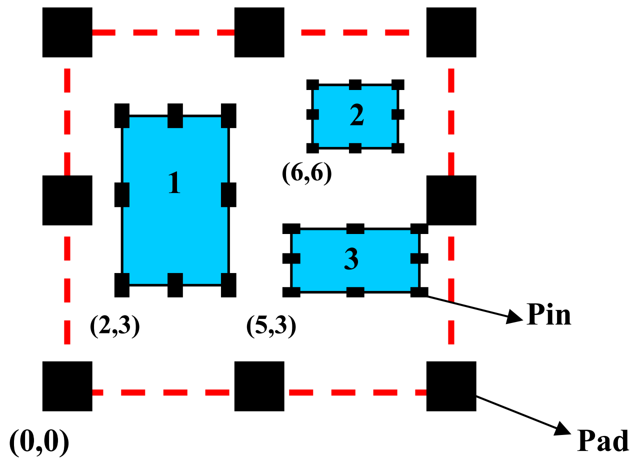


Figure 2.16: Block and terminal locations in a chip.

2 There are two types of terminals: terminals attached to the frame and terminals attached to the blocks. The first type is denoted as *pad* and the second as *pin* as shown in Figure 2.16.

3 Each block has a set of pins at locations specified by (x, y) co-ordinates as depicted in Figure 2.17. The process of identifying a pin location is called pin assignment. Note that the pin locations are determined relative to the chip's origin and are termed as absolute co-ordinates (x_{abs}, y_{abs}). Pin locations specified relative to block's lower left co-ordinates (x, y) are termed as relative co-ordinates (x_{rel}, y_{rel}). Relationship between absolute and relative pin co-ordinates is given by the following equations.

$$x_{abs} = x + x_{rel} \qquad (2.6)$$

and $$y_{abs} = y + y_{rel} \qquad (2.7)$$

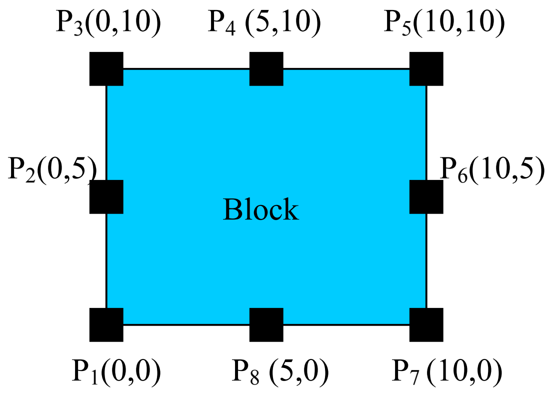


Figure 2.17: Pin locations on a block.

4 A block can be placed on a chip in any one of the four orientations as depicted in Figure 2.18. Note that the pins rotate too with the block rotation and their locations get modified accordingly.

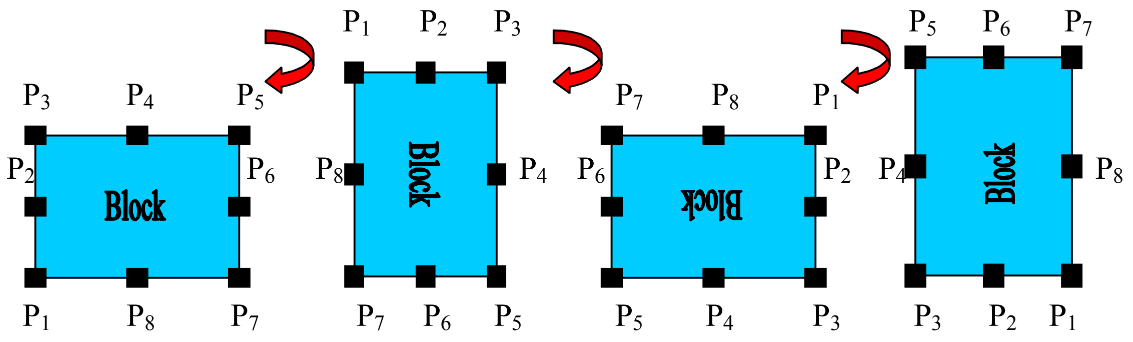


Figure 2.18: Different block orientations and corresponding pin locations.

5 A net is just a set of two or more pins. Figure 2.19 shows an example of a five terminal net where two pins are from block 3, other two from blocks 1 and 2 respectively, and the fourth terminal is a pad. Exact wirelength of each net is not known until routing is done.

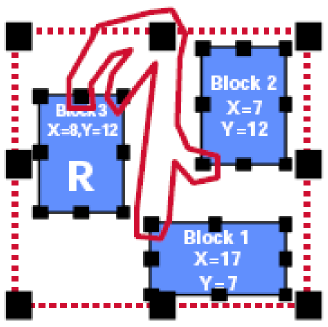


Figure 2.19: Example of a five terminal net.

2.1.2.2 Half Perimeter Wirelength Estimation (HPWL) method

There are many methods available to estimate wirelength in a floorplan. However, HPWL is the most popular method for wirelength estimation.

We compute the *HPWL* of a net containing *j* terminals, each having location (x_j, y_j) as follows:

$$HPWL_x = max\ (x_j) - min\ (x_j) \tag{2.8}$$

$$HPWL_y = max\ (y_j) - min\ (y_j) \quad (2.9)$$

$$HPWL = HPWL_x + HPWL_y \quad (2.10)$$

As shown in Figure 2.20, HPWL perimeter length metric for a two terminal net is determined by enclosing the net in a rectangle and computing the semi perimeter of the rectangle as below:

$$|\ rightX - leftX\ | + |topY - bottomY| \quad (2.11)$$

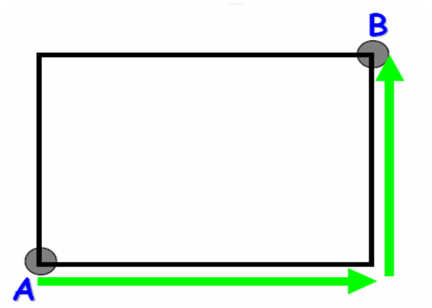
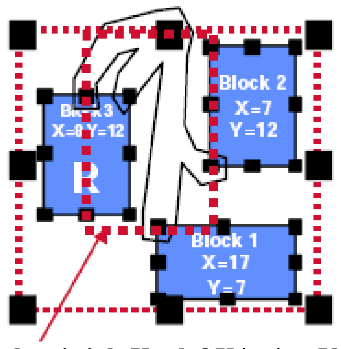


Figure 2.20: HPWL of a net.

We illustrate the determination of HPWL by an example of a five terminal net shown in Figure 2.19. The net consists of one chip pin i.e. pad and four other block pins. HPWL is determined by enclosing the net in a rectangle as shown dotted in Figure 2.21 below. The semi perimeter of the rectangle gives the HPWL of the net.



HPWL of the rectangle = | rightX − leftX | + |topY − bottomY|

Figure 2.21: Example of determination of HPWL of a five terminal net.

Table 2.2 lists the (*x, y*) co-ordinates for all the pins in the net. We calculate the value of

$HPWL_x$ and $HPWL_y$ from equations (2.8) and (2.9) respectively. Finally, we sum up $HPWL_x$ and $HPWL_y$ to yield the $HPWL$ of the net.

Table 2.2. Pin locations of a five terminal net.

Pin number	Pin co-ordinates (x, y)
1	(2, 7)
2	(4, 7)
3	(5, 3)
4	(6, 5)
5	(5, 10)

$HPWL_x = max\ (x_j) - min\ (x_j) = 6 - 2 = 4$

$HPWL_y = max\ (y_j) - min\ (y_j) = 10 - 3 = 7$

$HPWL = HPWL_x + HPWL_y = 4 + 7 = 11$

Total wirelength is given by the summation of lengths for all nets.

2.1.3 Cost Function

As the number of feasible solutions for a given instance of a floorplanning problem is very large, floorplanning algorithms use cost function as a measure that allows selecting superior floorplans with specific criteria. The possible criteria may be minimizing area, wirelength, delays, optimizing routing structure, power density and temperature of the chip or a combination of two or more of the above criteria. The specific criterion ensures greater reliability and performance of the circuits. A commonly used objective function is a weighted sum of area and wirelength:

$$Cost = \alpha \times A + \beta \times L \qquad (2.12)$$

where A is the total area of the packing, L is the total wirelength, and α and β are constants.

2.1.4 Floorplanning Algorithms

Several floorplanning algorithms exist that search over a solution space to determine the optimal floorplan solution. Floorplanning algorithms are classified into three classes:

1. **Constructive:** These algorithms attempt to build up a feasible solution by starting from a seed module; then, other modules are selected one at a time and added to the partial floorplan. Algorithms that fall under this category are as below.
    - Cluster Growth
    - Partitioning and Slicing
    - Mathematical Programming
    - Rectangular Dualization
2. **Knowledge-Based approach:** A knowledge expert system is implemented consisting of 3 basic elements.
    - Knowledge base containing data describing the floorplan problem and its current state.
    - Rules stating how to manipulate the data in the knowledge base in order to progress toward a solution.
    - Inference engine controlling the application of the rules to the knowledge base.
3. **Iterative**: These algorithms employ techniques that start from an initial floorplan which then undergoes a series of perturbations until a feasible floorplan is obtained or no more improvements can be achieved. Algorithms that fall under this category are enumerated as below.

- Simulated Annealing
- Force Directed Interchange
- Genetic Algorithm

Some of the most popular algorithms used in research are discussed in the section below.

2.1.4.1 Cluster Growth

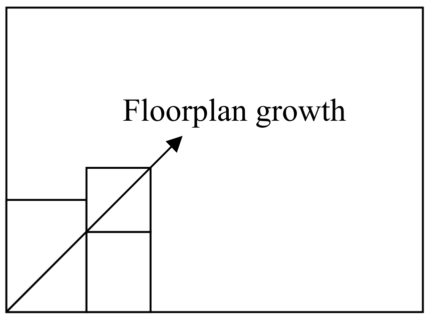


Figure 2.22: Cluster growth floorplanning.

In this approach, the floorplan is constructed in a greedy fashion; one module at a time until each module is assigned to a location of the floorplan. A seed module is selected and placed into a corner of the floorplan (lower left corner). Then, the remaining modules are selected one at a time and added to the partial floorplan while trying to grow evenly on upper, diagonal, and right sides simultaneously (Figure 2.22), maintaining any stated aspect ratio constraint on the modules as well as the chip itself and optimizing other criteria. Criteria might include: minimization of wiring length, minimization of dead space or both.

To determine the order in which the modules should be selected, the modules are initially organized into a linear order. Linear ordering algorithms order the given module netlist into a linear list so as to minimize the number of nets that will be cut by any vertical line drawn between any consecutive modules in the linear order. Linear ordering is one of the most widely used techniques for constructively building an initial placement configuration. A general description of a linear ordering algorithm is given in Figure 2.23, which is based on the linear ordering heuristic reported by Kang.

```
Algorithm Linear_Ordering
S: Set of all modules
Order: Sequence of ordered modules  // initially empty
Begin
    Seed = Select Seed module
    Order = [Seed]
    S = S – [Seed]
    Repeat
        For each module m Є S
            Compute the gain for selecting module m
            $gain_m$ = number of nets terminated by m – number of new nets started by m
        End For
        Select the module m* with maximum gain
        If there is a tie then
            Select the module that terminates the largest number of nets
        Elseif there is a tie then
            Select the module that has the largest number of continuing nets
        Elseif  there is a tie then
            Select the module with the least number of connections
        Else break remaining ties as desired
        Endif
        Order = [!Order, m*]  //append m* to the ordered sequence
        S = S − {m*}
    Until S = θ
End.
```

Figure 2.23: Linear ordering algorithm.

First, a seed module is selected. The seed selection could be random or based on the module connectivity with the I/O pads and/or the remaining modules. Then, the algorithm enters a Repeat loop. At each iteration of this loop, a gain function is computed for each module in the set of the remaining unordered modules. The module with the maximum gain is selected, removed from the set of unordered modules and added to the sequence of ordered modules. In case of a tie between several modules, the module which terminates the largest number of started nets is selected. In case of another tie, the module that is connected to the largest number of continuing nets is preferred. If we have one more tie, the most lightly connected module is selected. Remaining ties are broken as desired. The concept of net termination, starting of new nets and continuing nets are illustrated in Figure 2.24.

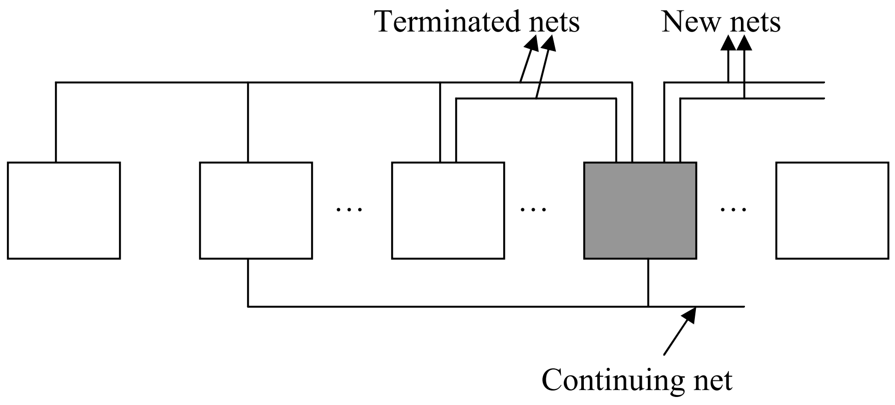


Figure 2.24: Classification of nets during linear ordering.

In the description of Figure 2.24, the notation !*L* is used to mean the elements of sequence *L*. Curly braces ({}) are used with sets and square brackets ([]) are employed with sequences. A general description of the cluster growth algorithm is given in Figure 2.25.

```
Algorithm Cluster_Growth
S: Set of all modules
Begin
    Order = Linear_Ordering(S)
    Repeat
        Nextmodule = b where Order = [b, !rest]
        Order = rest
        Select a location for b that will result in minimum increase in cost function
     // Cost may be function of the contour of the partial floorplan, size and shape of
        b, and wiring length
    Until Order = θ
End.
```

Figure 2.25: Cluster growth algorithm.

2.1.4.2 Genetic Algorithms

Genetic algorithms (GA) [3] are a class of search and optimization methods that mimic the evolutionary principles in natural selection. They are implemented as a computer simulation, in which a population of abstract representations (called chromosomes) of candidate solutions (called individuals) to an optimization problem evolves towards better solutions. Genetic algorithms are applied to floorplanning problem to search for an optimized solution from an infinitely large solution space. Figure 2.26 shows a genetic algorithm optimization flow. The solution (i.e. representation of a floorplan design) is usually encoded into a binary string called chromosome. Instead of working with a single solution, the search begins with a random set of chromosomes (floorplans) called initial population. Each chromosome is assigned a fitness score that is directly related to the objective function of the optimization problem.

The population of chromosomes (floorplans) is modified to a new generation by applying three operators similar to natural selection operators – *Reproduction*, *Crossover* and *Mutation*. *Reproduction* selects good chromosomes based on the fitness function and duplicates them. *Crossover* picks two chromosomes randomly and some portions of the chromosomes are exchanged with a probability P_c. Finally, *mutation* operator changes a 1 to a 0 and vice versa with a small mutation probability P_m. A genetic algorithm successively applies these three operators in each generation until a termination criterion is met. It can very effectively search a large solution space while ignoring regions of the space that are not useful. This algorithmic methodology leads to very time-efficient searches. In general, a genetic algorithm has the following steps:

1. Generation of initial population.

2. Fitness function evaluation.

3. Selection of chromosome.

4. Reproduction, Crossover and Mutation operations.

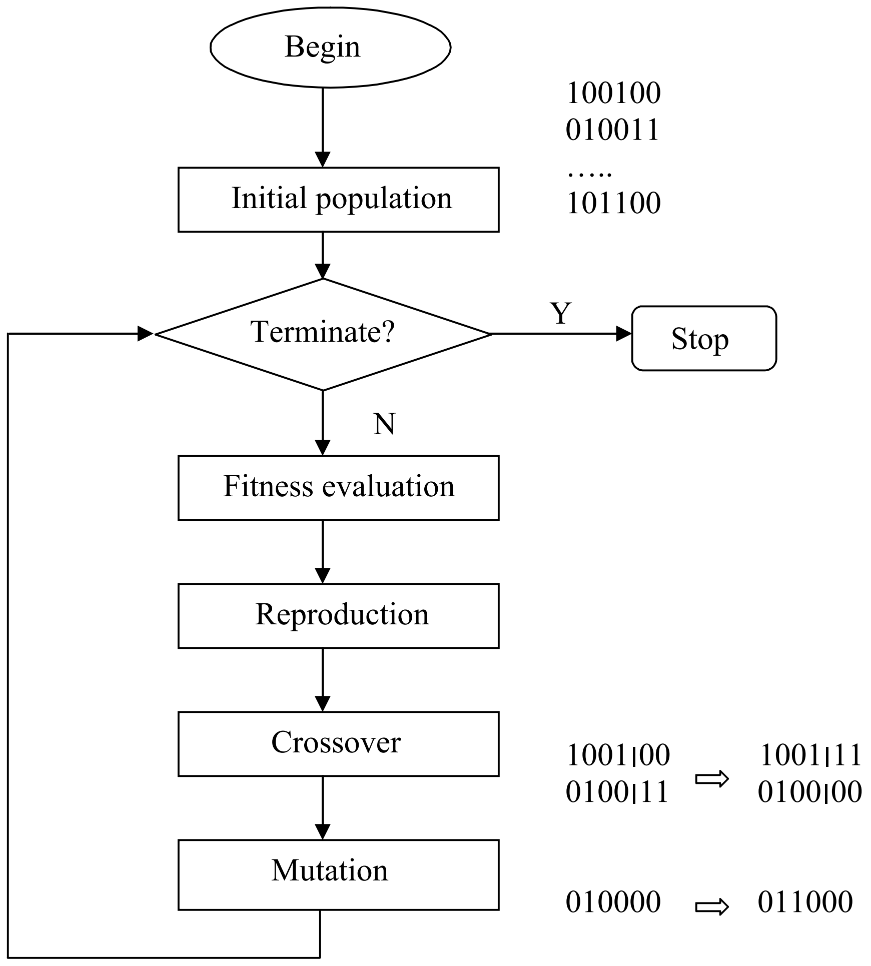


Figure 2.26: Genetic algorithm flow.

2.1.4.3 Simulated Annealing

It is a technique to find a good solution to an optimization problem by trying random variations of the current solution. A worse variation is accepted as the new solution with a probability that decreases as the computation proceeds. The slower the cooling schedule or rate of decrease, the more likely the algorithm is to find an optimal or near-optimal solution. An annealing algorithm needs four basic components:

- **Solution space (e.g. slicing floorplan):** These represent the possible problem solutions over which we will search for an answer.
- **Perturbation rules:** A set of allowable moves that will permit us to move from one feasible configuration to other as annealing proceeds.
- **Cost function (e.g. area):** Determines how "good" a particular solution is.
- **Cooling schedule:** To anneal the problem from a random solution to a good, frozen placement.

We need a starting hot temperature and rules to determine when the current temperature should be lowered, by how much the temperature should be lowered, a cooling schedule (e.g. $T = 0.9 * T$) and when annealing should be terminated. The pseudo code for the simulated annealing algorithm with reference to floorplan problem is outlined in Figure 2.27.

The simulated annealing algorithm starts by randomly choosing an initial B*-Tree. Then, it perturbs a B*-Tree (a feasible solution) to another B*-Tree based on the aforementioned Op1–Op3. The perturbation process converts one feasible B*-Tree to another. We then do the placement for the corresponding B*-Tree and evaluate the cost function. The move is accepted if the cost of the current solution is less than the previous one or with a probability that is a decreasing function of annealing temperature (Boltzmann function) as defined in the algorithm. For all other cases, the move is rejected. At each temperature, we try enough moves until there are N uphill moves (bad moves) or the total number of moves exceeds $2N$ where N is an increasing function of n ($O(n)$), the number of basic rectangles. We use a fixed ratio temperature schedule i.e. $T = 0.9 * T$. We terminate the annealing process if the number of accepted moves is less than 5% of all moves made at a certain temperature or the temperature is low enough i.e. less than the threshold. At last, we transform the resulting B*-Tree, i.e. the solution with the lowest value of cost function, to the corresponding final admissible placement.

```
Begin
    Initialize temperature T
    Initialize a B*-Tree for the input blocks
    Do
        Repeat
            Perturb
            Placement
            Compute cost
            If cost < Previous cost
                Accept the move
            Else
                Prob = min (1, $e^{-\Delta ck/T}$), where $\Delta_c$ = change in cost, k = constant
                Rand = Random (0, 1)
                If Rand $\leq$ Prob then
                    Accept the move
                Else
                    Reject the move
                End if
            End if
        Until (Uphill moves > 2*N or Downhill moves < N)
        T = 0.9 * T
    While (T $\geq$ Threshold and Reject_rate $\leq$ converge_rate)
End.
```

Figure 2.27: Pseudo code of simulated annealing algorithm.

2.1.5 Floorplanning Methodology

A floorplanning problem typically consists of an input as a benchmark suite that consists of standard circuits (set of blocks with dimensions of each block, possible shapes of each block, number of terminals for each block) with netlist that can be tested upon to obtain an optimized floorplan.

We have to make certain selections, as given below, in floorplanning approach to obtain the output which is a layout with orientations and positions of blocks.

1. **Choice of floorplan representation**: We select an appropriate floorplan representation according to the input as well as implementation cost and run time.
2. **Choice of cost function**: Cost function contains parameters that determine the criteria for selection of an optimized floorplan. E.g. area, wirelength, power etc.
3. **Choice of floorplanning algorithm**.

Selection of all these parameters will determine how good the optimized floorplan solution is.

2.2 Thermal and Power Dissipation effects

Smaller feature size, higher packing density and rising power consumption lead to dramatic temperature increase in modern high performance very large scale integrated (VLSI) circuits, thereby resulting in serious timing and reliability concerns. Therefore, it becomes important to identify the major power dissipation sources on a chip and quantify them accurately. Both the device and interconnect power dissipation contribute towards the estimation of the resulting chip temperature. This enables to optimize the on chip thermal distribution profile, thereby eliminating hotspots.

2.2.1 Sources of Power Dissipation

Generally there are two sources of power dissipation in a chip:

1) *Static power dissipation*, which is switching independent and mostly induced by various short-circuit and leakage currents.

2) *Dynamic power dissipation*, which arises from the switching activities of

logic circuits.

We take into account both static and dynamic sources of power dissipation while calculating the power density of functional blocks and interconnects in the design.

2.2.1.1 Dynamic Power Dissipation

Dynamic power is the main source of power consumption on a chip at algorithm and architecture levels. This is because short-circuit and leakage currents responsible for static power,

(i) Can be reduced to less than 15% of the total chip power by smart circuit design techniques [41], and

(ii) Are influenced mainly by the circuit design style used.

Dynamic power dissipation mainly arises from two circuit behaviors:

1) Transient short-circuit current, and
2) Repeated charging and discharging of capacitive loads.

The short-circuit current is incurred due to transient conduction of both the pull-up and pull-down circuits in the CMOS circuit. Because transition cannot realistically be instant, it is possible that the shut-off network is turned on before the previously turned-on network is shut off. This current, however, is not significant in most circuits and is often ignored [27] and [28].

The major dynamic power consumption comes from the charging and discharging of the state-keeping nodes. A low-to-high state transition corresponds to the charging up of all the capacitors associated with that node; while a high-to-low transition corresponds to the discharging of the node. With scaled feature sizes, the capacitance per unit area increases, accompanied by the increased switching frequency. These trends lead to significant dynamic power consumption in modern-day circuits.

The dynamic power dissipated is given by

$$P_{dynamic} = \alpha \times C \times V^2 \times F \tag{2.13}$$

where α is the switching activity, F is the frequency of operation, V is the supply voltage and C is the physical capacitance of the resources.

2.2.1.1.1 Interconnect Power Dissipation

Assuming the supply voltage and frequency of operation to be constant, interconnect power dissipation depends on the switching activity and the capacitance of the interconnects. The capacitance of the interconnect C_{int} is directly dependent on its length L_{int} which is determined by the floorplan. Thus

$$C_{int} = k \times L_{int}$$

where k is some constant. The switching activity depends on the number of times the interconnect is accessed and the correlations between the data that it operates on. The access in turn depends on the total number of data transfers in the algorithm while the correlations depend heavily on the input data [26]. In other words, the switching activity depends on how frequently a particular interconnect is used.

For example, two functional blocks that need to access each other very frequently for data or information exchange will have their bus interconnects busy all the time, leading to a greater current flow. This would lead to higher power dissipation compared to when the interconnect is used less frequently. The lesser the interconnect is used, lesser is the power consumption as the bus will be in high impedance state most of the time. Such type of impact will be more pronounced in high frequency circuits. However, at the floorplanning stage, we can safely assume that the switching activity depends mainly on the number of accesses to interconnects, which in other words is the frequency of usage of interconnects.

Thus, to minimize the interconnect power dissipation, it is important to consider both the switching activity and the net length with the objective of minimizing the length of interconnects, which have high switching activity, in the resulting floorplan. Hence, interconnects which are heavily accessed or have high switching activity on them should be made as short as possible to reduce the power dissipation and the wire delays.

2.2.2 Temperature Estimation

The temperature of the chip depends on the power consumption of each functional block and the relative positions of the functional blocks. Also, the temperature greatly depends on the power density profile of the chip, i.e. how the total power is distributed among the various blocks on the chip. For the same total value of power dissipation and the same floorplan layout, different distribution of power among various blocks can lead to altogether different temperature profiles of the chip. This is due to the fact that heat transfer depends on adjacent hot and cold blocks. Since high power blocks generate more heat, placing them adjacent to low power density or colder blocks will lead to larger heat diffusion than placing them closer to hot blocks. This will result in more spreading of heat, thus reducing hotspot temperatures. The larger the temperature difference, the larger the heat diffusion. Therefore, to reduce the maximum temperature of the chip, we should surround blocks with high power density by blocks with low power density if possible.

We explain the impact of power density profile on heat diffusion with the help of an example. Let there be four blocks with total power of 10 units. Let the power distribution profile for one case be as given on the next page. The corresponding heat diffusion flow is indicated besides the power distribution profile.

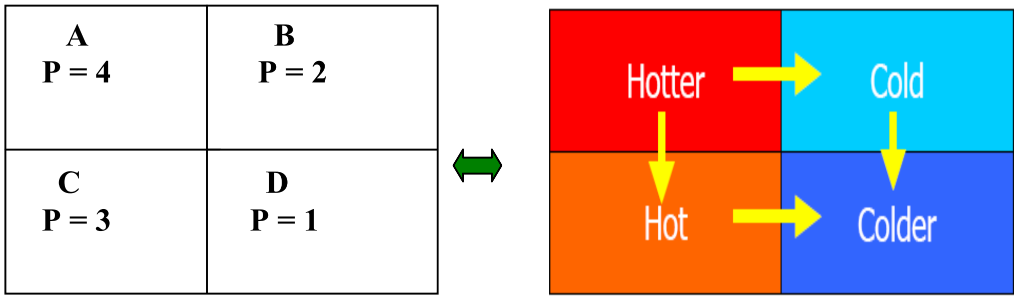


Figure 2.28: Case 1: a) Power distribution profile b) Corresponding heat diffusion flow.

On changing the power density profile, as in case 2, we get a different corresponding heat diffusion flow as given below.

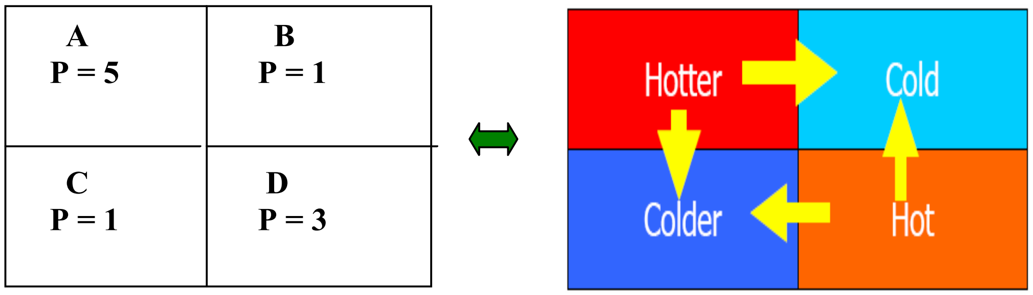


Figure 2.29: Case 2: a) Power distribution profile b) Corresponding heat diffusion flow.

According to the general temperature-power equation

$$T = P \times R \qquad (2.14)$$

where P is the power and R is the thermal resistance. Substituting the value of R, the equation modifies to as given below

$$T = P \times (t / k \times A) = (P / A) \times (t / k)$$

where t is the thickness of the chip, A is the area and k is the thermal conductivity of the material. The equation can now be written as below

$$T = d \times (t / k) \qquad (2.15)$$

where *d* is the power density. Based on previous equation, we can conclude that temperature is heavily dependent on power density. Thus, we can substitute power density metric for temperature in thermal-aware calculations.

2.2.2.1 Hotspot Tool

Skadron et al. [29] proposed a thermal modeling tool called HotSpot which is easy to use and computationally efficient for modeling thermal effects at the block level. Hotspot provides a simple compact model where the heat dissipation within each functional block and the heat flow among blocks are accounted for. An RC network of thermal capacitances and resistances of functional modules are constructed and then temperatures at the center of functional modules are calculated by using circuit-solving techniques. The basic idea is that, we define the transfer thermal resistance R_{ij} of functional block *i* with respect to block *j* as the temperature rise at block *i* due to one unit of power dissipated at block *j*:

$$R_{ij} = \Delta\, T_{ij} / \Delta\, P_j$$

such that we can get a transfer thermal resistance matrix as $\mathbf{R^t}$ **.** For any power distribution on the floorplan, we can calculate each block's temperature by using the following equation:

$$\begin{bmatrix} \mathbf{T_1} \\ \mathbf{T_2} \\ \vdots \\ \mathbf{T_m} \end{bmatrix} = \begin{bmatrix} \mathbf{R^t_{11}} & \mathbf{R^t_{12}} & \cdots & \mathbf{R^t_{1m}} \\ \mathbf{R^t_{21}} & \mathbf{R^t_{22}} & \cdots & \mathbf{R^t_{2m}} \\ \vdots & & \ddots & \vdots \\ \mathbf{R^t_{m1}} & \mathbf{R^t_{m2}} & \cdots & \mathbf{R^t_{mm}} \end{bmatrix} \begin{bmatrix} \mathbf{P_1} \\ \mathbf{P_2} \\ \vdots \\ \mathbf{P_m} \end{bmatrix}$$

Figure 2.30: Transfer thermal resistance matrix $\mathbf{R^t}$.

where P_i is the power consumption and T_i is the temperature of the functional block *i.* The transfer thermal resistance matrix can be obtained from Hotspot, given the floorplan for a set of blocks.

The inputs to HotSpot are the floorplan description and the power consumption number of individual modules. The specifications of heat spreader and heat sink are also provided to define the heat-removing ability. The output of Hotspot is the temperature for each module.

2.3 Variation Models

Design uncertainty greatly affects dimensions of blocks and interconnect lengths in complex VLSI circuits at an early chip planning stage. There are many sources of uncertainty, chief among them being: incomplete design of some blocks and incomplete technology library of cells. The resulting variations affect other chip parameter estimations like area, wirelength, delay and power etc, which need to be assessed for the acceptability of chip architecture at the design decision stage.

We need methods and tools to model arbitrary variational CAD problems. Monte Carlo analysis remains the gold standard for "arbitrary" problems- accurate, but often intractably slow. To model the variations accurately in less time, we use some analytical and statistical methods. Since we take into account the variations in dimensions of blocks and interconnect lengths to determine the range of chip area and wirelength, we need to perform arithmetic calculations on interval ranges. To facilitate such range computations, we resort to the use of some kind of numerical computation methods.

Numerical Computations is the study of algorithms for the problems of continuous mathematics. Many numeric computations are inherently approximate, i.e. they will not deliver the "true" exact values of target quantities but only some values in some sense "near" the true ones. The difference between a computed value and the "true" value of the corresponding quantity is called the error of that computed value. In Self Validated Computation (SVC), the accuracy of computed quantities is being tracked as part of the process of computing them. So, if the magnitude of the error cannot be predicted, at least it can be known a posteriori.

Interval arithmetic (IA) and affine arithmetic (AA) are two such SVC models based on range analysis, i.e. use ranges to approximate the accurate values. Whatever the shape of allowed ranges, all range-based SVC models provide, for every function $f: R^m \rightarrow R^n$, a range extension $F: R^m \rightarrow R^n$ with the following property: If the input vector $(x_1, \ldots, x_m)$ lies in the range jointly determined by the given approximate values $X_1, \ldots, X_m$, then the quantities $(z_1, \ldots, z_n) = f(x_1, \ldots, x_m)$ are guaranteed to lie in the range jointly determined by approximate values $(Z_1, \ldots, Z_n) = F(X_1, \ldots, X_m)$.

The IEEE Floating-point Standard provides control over rounding, a feature that is essential to SVC. The standard specifies precisely the results of exceptional operations and reserves certain bit patterns to denote two "infinite" values and a series of error codes or "not-a-numbers" (NaN).

- We use the notation $< f >$ for the value of expression f in IEEE floating-point arithmetic with the default rounding mode.
- We write $\uparrow f \uparrow$ for a numerical float (possibly) that is greater than or equal to the value of a formula f; that is, the value of f is rounded up to a representable number (not necessarily the smallest one). Similarly, we write $\downarrow f \downarrow$ for the value of f rounded down to a representable number.

- In special cases, f can consist of a single, or two or more arithmetic operations.

2.3.1 Interval Arithmetic (IA)

Interval arithmetic (IA) [36] is a range-based model of numerical computation. In IA, each real quantity x is represented by an interval $x = [x_{lo}, x_{hi}]$ of real numbers. It means that the true value of x is known to satisfy $x_{lo} \leq x \leq x_{hi}$.

(1) We define a non-empty interval as a set of this form:

$[x_{lo}, x_{hi}] = \{x \in R\text{: } x_{lo} \leq x \leq x_{hi}\}$,

where x_{lo} is in $F \cup \{-\infty\}$ and x_{hi} is in $F \cup \{+\infty\}$.

We define an empty interval [ ] where the lower and upper bounds are not defined.

(2) The bounds of an interval are float values, possibly infinite, but its elements are finite real quantities.

(3) A finite float x can be represented as an interval $[x_{lo}, x_{hi}]$.

(4) The pairs: $[+\infty, +\infty]$, $[-\infty, -\infty]$, [NaN, NaN], [a, NaN], [NaN, a] are not valid, for any float a.

Operations are defined on intervals e.g. negation, addition, translation, subtraction, scaling, multiplication, reciprocal, division, square root, logarithm, exponential and sine and co-sine etc. Other operations are midpoint, radius, meet (intersection), join (convex hull). We present two examples of affine operations, namely addition and join below.

Example 1: Addition

```
IA.add ($x$, $y$: interval): Interval ≡   | Computes x + y.
if $x$ = [ ] or $y$ = [ ]      then return [ ]
else   return [$\downarrow x_{lo} + y_{lo} \downarrow$, $\uparrow x_{hi} + y_{hi} \uparrow$ ]
```

Example 2: Join

```
IA.join (x, y: interval): Interval ≡   | Returns x U y.
if $x$ = [ ]  then return $y$
else if $y$ = [ ]  then return $x$
else  return [min{$x_{lo}$, $y_{lo}$}, max{$x_{hi}$, $y_{hi}$}]
```

The main weakness of IA is its over-conservatism: the computed interval for a quantity may be much wider than the exact range of that quantity. In long computation chains, the relative accuracy of the computed interval decreases at an exponential rate and finally the "error explosion" occurs. Some techniques to avoid error explosion are:

1. Avoid unfavorable correlations between the arguments of the IA operation.

   Example: x = [4, 6], considering the evaluation of $z \leftarrow x \times (10 - x)$

   Using IA.sub and IA.mul, we get z = [16, 36]. However, a trivial analysis shows that the exact range should be [24, 25].

   So, the correlation between the arguments is unfavorable.

2. Combine several arithmetic operations into a single "macro operation" and write a special-purpose IA routine for it.

   Example: x = [-2, 2], considering the evaluation $z \leftarrow x^2$. Since $x^2 = x \times x$, so we use IA.mul and we get [-4, 4] which is of poor accuracy.

   So, we can write a special routine (IA.sqr, for example) to deal with evaluation of power.

Unfortunately, the previous techniques can only be applied to relatively simple operations over restricted domains. When the expression to be computed is determined only at run time or involves dozens of variables and operations, avoiding bad correlation is impossible. We have to resort to more sophisticated SVC models, for example, affine arithmetic (AA).

2.3.2 Affine Arithmetic (AA)

Affine arithmetic (AA) [37] is introduced to overcome the error explosion problem occurred in IA. It provides a tighter bound for the computed quantities. In affine arithmetic, each input or computed quantity x is represented by an affine form $\hat{x}$ which is a first order polynomial

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \ldots + x_n\varepsilon_n \qquad (2.16)$$

where x_0, x_1, x_2, …, x_n are known floating-point numbers, and ε_1, ε_2, ..., ε_n are symbolic variables whose values are only known to lie in the range [-1,+1]. x_0 is the ideal value of the affine form . Each ε_j stands for an independent component of the total uncertainty. x_j gives the corresponding magnitude of component ε_j. It means, if we want to evaluate the range of $Z = x \times y$ ($x \in [a, b]$, $y \in [c, d]$), we will have to replace x and y with

$$Z = (x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \ldots + x_n\varepsilon_n) \times (y_0 + y_1\varepsilon_1 + y_2\varepsilon_2 + \ldots + y_n\varepsilon_n)$$

where x_j is determined by a and b, and y_j by c and d.

The approximation error incurred in each AA operation normally has a quadratic dependency on the size of the input intervals. Therefore, if the input intervals are small enough, each operation will provide a fairly tight estimate of the exact range of the corresponding quantity.

Addition, subtraction and simple scaling are easily seen to yield the affine form directly. In affine operations, for a function, if $f(x, y)$ is an affine function of x and y, namely $f(x, y) = \alpha x + \beta y + \zeta$, then z can be represented by an affine form directly,

$$\hat{Z} <= \alpha\hat{x} + \beta\hat{y} + \zeta = (\alpha x_0 + \beta y_0 + \zeta) + (\alpha x_1 + \beta y_1) \times \varepsilon_1 + \ldots + (\alpha x_n + \beta y_n) \times \varepsilon_n$$

This will yield an almost-exact range (except for round-off error) of z in terms of ε_j in the input range.

The general rule for approximating the result of a non-affine operation (e.g. x, /, exp) on affine operands is to seek an affine form that is a linear combination of the operands along with a new term (ε) to account for the error. The range of the actual result should lie within the range of this affine approximation. In non-affine operations, if in $z = f(x, y)$, f can not be represented by an affine form, then we need to add the approximation error, namely

$$\hat{Z} = f^a(e_1, \ldots, e_n) + z_k\varepsilon_k = z_0 + z_1\varepsilon_1 + \ldots z_n\varepsilon_n + z_k\varepsilon_k$$

where f^a is the affine operation and $z_k\varepsilon_k$ represents the approximation error as well as the round-off error.

Round-off errors can not be avoided, so in order to provide guaranteed enclosure, every affine form should add an extra term $z_k\varepsilon_k$. E.g. $Z = X + Y$ should add the extra term $z_k\varepsilon_k$. The handling of round-off errors increases the code complexity and execution time of AA operations. In applications where those errors are known to be unimportant (because they are dominated by uncertainties in the input data, etc), round-off error control can be ignored.

We explain the non-affine operation with the help of an example. Let's consider a multiplication operation $\hat{z} = \hat{x} \times \hat{y}$

where $\hat{x} = 30 - 4\varepsilon_1 + 2\varepsilon_2$ and

$\hat{y} = 20 + 1\varepsilon_1 + 3\varepsilon_2$, then

$\hat{z} = 600 + 10\varepsilon_1 + 40\varepsilon_2 + 30\varepsilon_3 + (-4\varepsilon_1 + 2\varepsilon_2)(3\varepsilon_1 + 1\varepsilon_3)$

The quadratic term $(-4\varepsilon_1 + 2\varepsilon_2)(3\varepsilon_1 + 1\varepsilon_3)$ can be treated as the error term whose range is [-24, 24] writ large. Hence, $\hat{z}$ is in the range of 600 $\pm$ 104 = [496, 704]. Analysis shows that the actual range of $\hat{z}$ is [528, 675], so AA results in only 1.42 times wider than the actual range. If IA is used, the resulting range is [384, 863], 3.26 times wider than the actual range.

Thus, AA gives more precise estimation than IA and is a preferred method in range calculations where accuracy is essential. To find the best estimation, Chebyshev approximation can be used.

To convert from AA to IA, we define the smallest interval that contains all possible values of $\hat{x}$. Every affine form $\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \ldots + x_n\varepsilon_n$ implies a bound of the ideal quantity x, namely r, i.e.

$$x \in x = [x_0 - r, x_0 + r] \tag{2.17}$$

where r is the total deviation of $\hat{x}$, $\sum_{j=1}^{N} (x_j)$. It is the smallest interval that contains all possible values of $\hat{x}$. However, this conversion discards all correlation information in $\hat{x}$.

To convert from IA to AA, we replace every ordinary interval bound $x = [a, b]$ for an ideal quantity x by $\hat{x} = x_0 + x_k\varepsilon_k$ where

$$x_0 = (a + b) / 2\ , \tag{2.18}$$

$$x_k = (b - a) / 2 \tag{2.19}$$

and ε_k a new noise symbol not occurred anywhere before.

2.3.3 Monte Carlo Simulation

Monte Carlo simulation [39] is a method for iteratively evaluating a deterministic model using sets of random numbers as inputs. This method is often used when the model is complex, nonlinear or involves more than just a couple uncertain parameters. It is used for analyzing uncertainty propagation where the goal is to determine how random variation, lack of knowledge or error affects the sensitivity, performance or reliability of the system that is being modeled. Monte Carlo simulation is categorized as a sampling method because the inputs are randomly generated from probability distributions to simulate the process of sampling from an actual population. So, we try to choose a distribution for the inputs that most closely matches data we already have or best represents our current state of knowledge. The data generated from the simulation can be represented as probability distributions (or histograms) or converted to error bars, reliability predictions, tolerance zones and confidence intervals. A simulation can typically involve over 10,000 evaluations of the model.

Monte Carlo simulation method is generally used as a base for comparison with the proposed algorithm to verify the accuracy and effectiveness of the algorithm to evaluate the deterministic model by some other approach.

Chapter 3

Interconnect Power and Thermal aware Floorplanning

Thermal aware floorplanning has become of paramount importance recently due to scaling technologies and many more research efforts are directed towards determining accurate temperature estimates of the chip. Since thermal aware floorplanning requires knowledge of both the power density profile and the relative positions of blocks, accurate estimations of both the factors are essential. In our work, we seek to do accurate thermal conscious floorplanning by considering the above factors with in-depth analysis. In this chapter, we describe our interconnect power and thermal aware floorplanning algorithm based on accurate power estimations by taking into account the effect of switching activity of interconnects on the power dissipation and hence the temperature estimations of the chip.

The chapter is organized as follows. Section 3.1 reviews previous work related to this research. Section 3.2 formulates the problem. We present our approach in evaluating the impact of interconnects during floorplanning process in section 3.3. Section 3.4 presents and discusses experimental results. We conclude this chapter in the last section.

3.1 Motivation

In this section, we discuss some relevant work that has been a source of motivation for the research work presented in this chapter. Previous works related to this work fall into two broad categories- the first is the prevalence of thermal aware floorplanners and thermal modeling tools, and the second is the interconnect power estimates.

The first category of work studied thermal or temperature aware floorplanning algorithms at the micro architectural level. For example, Han et al. [15] used an Alpha floorplan to analyze the impact of floorplanning on the maximum temperature. The tradeoff between performance and temperature is explored in [14]. W.L. Hung [16] used the genetic algorithm approach. These papers consider that the power density can increase due to the placement of blocks, that have high power consumption, close together. However, all the above work neglected the interconnect power consumption. Tools for modeling thermal effects on chip-level placement have been developed [10], [17], [18] and [19]. Nevertheless, interconnect power factor is never the center of attention in these floorplanning/placement techniques.

Recently, W.L. Hung [20] developed a floorplanner that considers the interconnect power consumption in exploring a thermal-aware floorplan. However, the drawback of their approach is that they have only considered the interconnect lengths in calculating the distribution of total interconnect power across the chip. They have not accounted for the switching activity of the interconnects in the power density formulations, which can lead to erroneous chip temperature estimates. Our work uses a similar approach to theirs but also takes into account the effect of switching activity which has a significant impact on the power density profile of the chip and thus the average and peak temperatures.

The second category of related work studied interconnect effects. Interconnect buffers are now first-order timing and power considerations in VLSI design [21]. This change has imposed challenges across all design levels. It is no longer possible to accurately produce the power consumption and performance of a design without prior knowledge about its floorplan to predict the structure of its interconnect. A number of researchers have considered the impacts of chip-level interconnect in power and performance aspects [22], [23] and [24]. There has been significant work done at high-level synthesis stage. Prabhakaran [25] presented a new algorithm that combines physical design and high-level synthesis with the objective of minimizing interconnect energy dissipation. Mehra [26] considered architectural synthesis and power reduction techniques. However, none of the above papers consider the thermal or temperature effects. Our work focuses mainly on thermal effects in addition to the power dissipation at floorplanning stage.

There are numerous works on the determination of device power dissipation with accurate analysis. However, interconnect power dissipation is still an ongoing area of research with the sharp rising impact of interconnects on power dissipation of the chip with scaling technologies as discussed in chapter 1.2. Therefore, we mainly focus on interconnect power dissipation in our work and seek to accurately model it, considering all the major factors like switching activity of interconnects which has been ignored till now at the floorplanning stage. We then intend to do interconnect power and thermal aware floorplanning based on accurate determination of power dissipation.

3.2 Problem Formulation

We define our problem in this work as follows: given the information of a set of modules including their areas, interconnections and power consumptions, the interconnect power and thermal aware floorplanning problem is that of placing the modules in the chip area satisfying a set of conditions and achieving the goal of distributing the temperature evenly across the chip by taking into account the power dissipation due to switching activity of interconnects while optimizing area and wirelength. In this work, we consider only the hard modules, i.e. modules that are not flexible in shape but are free to move and rotate.

Let $B = \{b_1, b_2, \ldots, b_m\}$ be a set of m rectangular modules with block b_i of width W_i, height H_i, area A_i, and an original power density P_i, $1 \leq i \leq m$. Each module is free to rotate. Let (x_i, y_i) denote the co-ordinates of the bottom-left corner of the rectangle b_i on a chip. A floorplan F is an assignment of (x_i, y_i) for each b_i such that no two modules overlap. The goal of interconnect power and thermal aware floorplanning algorithm is to minimize

(i) Chip area (i.e. minimum bounding rectangle of F).

(ii) Wirelength (i.e. the summation of half bounding box of interconnections) and

(iii) Both peak and average temperatures across the chip.

3.3 Interconnect Power and Thermal aware Floorplanning Algorithm

In this section, we describe our methodology to develop the interconnect power and thermal aware floorplanning algorithm by taking into account the effect of switching activity of interconnects on the power dissipation and hence the temperature estimations of the chip.

3.3.1 Methodology and Algorithm

We first developed a traditional simulated annealing floorplanning algorithm having the cost function (equation 2.12)

$$C = \alpha \times A + \beta \times WL$$

where A is the total area of the packing, WL is the total wirelength, and α and β are constants which denote the relative weights of area and wirelength respectively in the cost function, $(\alpha + \beta) \leq 1$. We use B*-Tree representation for our floorplanning algorithm as its easy to implement, has a smaller solution space and time complexity, and many other advantages as outlined in chapter 2.1.1.4. Since our floorplanner considers only the hard modules, we perturb the B*-Tree (a feasible solution) to another B*-Tree by using the following three operations

Op1: Rotate a module.

Op2: Move a module to another place.

Op3: Swap two modules.

We obtained a floorplan description which gives the positions and dimensions of various functional blocks together with the length of different interconnects in the netlist. We then calculated the power density values for different modules by taking into account the interconnect induced power consumptions with and without the considerations of switching activity of the interconnects. Hotspot tool was then used to determine the temperature profile of the chip based on floorplan description and power density values calculated for various blocks as the inputs to the tool. But Hotspot has a limitation that it models thermal effects only at the per module level. It does not model the thermal effects arising due to interconnect power dissipation directly. So, we devised a mechanism similar to [20] to distribute the power consumed by each net to the connecting modules as explained in the next section on interconnect power distribution. Finally, we make our floorplanner interconnect power and thermal aware by modifying the cost function as discussed in the section on temperature approximations. Figure 3.1 on the next page summarizes the flow of our algorithm.

```
Input: A set of modules $b_i$ ($W_i$, $H_i$, $A_i$ and $P_i$), Netlist N and total interconnect power TIP.
Output: Floorplan F of chip area (A), total wirelength (WL), average and peak temperatures ($T_{avg}$ and $T_{peak}$), and containing modules with locations ($x_i$, $y_i$).
Begin
      Initialize temperature T
      Initialize a B*-Tree for the input blocks
      For each net $j \in N$
         $\alpha[j]$ = Random number $\in (0, 1)$
      End For
    // Run B*-Tree based simulated annealing floorplanning algorithm to obtain F
      Do
        Repeat
           Perturb
           Placement
           Call Calculate_Power algorithm to determine total block power $TP_i$
           Compute cost
           If cost < Previous cost
              Accept the move
           Else
              Prob = min (1, $e^{-\Delta ck/T}$), where $\Delta_c$ = change in cost, k = constant
              Rand = Random (0, 1)
              If Rand $\leq$ Prob then
                 Accept the move
              Else
                 Reject the move
              End if
           End if
        Until (Uphill moves > 2*N or Downhill moves < N)
        T = 0.9 * T
     While (T $\geq$ Threshold and Reject_rate $\leq$ Converge_rate)
     Apply F and $TP_i$ to Hotspot tool to obtain $T_{avg}$ and $T_{peak}$
End.
```

Figure 3.1: Pseudo code of the interconnect power and thermal aware floorplanning algorithm.

Figure 3.2 shows the diagram depicting the relationship between temperature profile and the effect of switching activity of interconnects on power density profile of the chip.

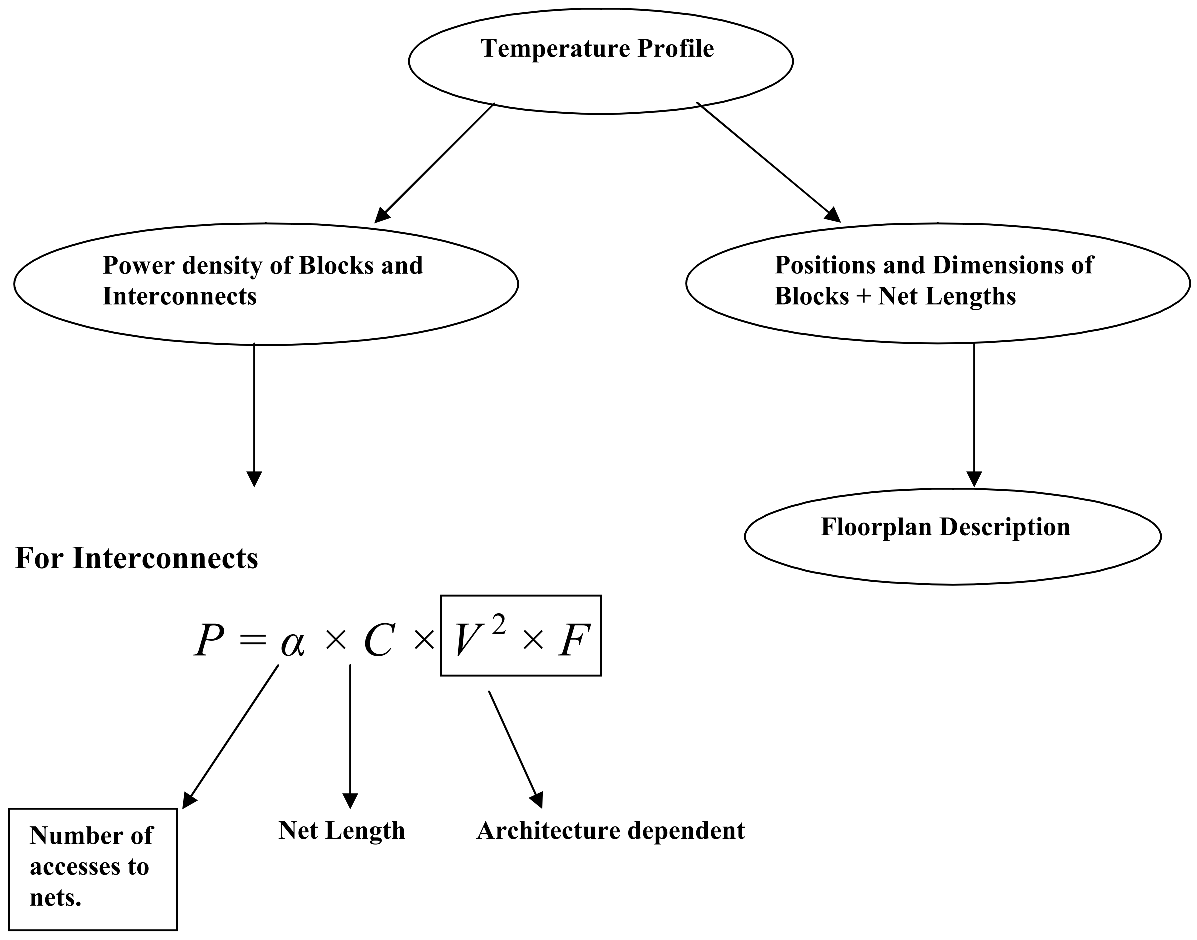

Temperature Profile
Power density of Blocks and Interconnects
Positions and Dimensions of Blocks + Net Lengths
Floorplan Description
For Interconnects
$P = \alpha \times C \times \boxed{V^2 \times F}$
Number of accesses to nets.
Net Length
Architecture dependent

Figure 3.2: Diagram showing relationship between temperature and switching activity of interconnect.

3.3.2 Interconnect Power Distribution

To account for the interconnect induced power consumptions in the Hotspot thermal modeling tool, we distribute the power consumed by each interconnect or net to the connecting modules in the floorplan. We accomplish this goal by taking the intuition that power consumption of a module is relative to capacitance and capacitance is proportional to the module area [20]. We thus distribute the net power to a connecting module in proportion of its module area and total area of all the modules connected in a net. Given a particular value of total interconnect power, we then calculate the proportion of total interconnect power assigned to each net with and without considering the effect of switching activity of nets. As outlined in section 2.2 earlier, interconnect power dissipation depends on both the net length and switching activity of interconnects. The switching activity of interconnect depends on its frequency of usage which is further dependent on the specific design and the internal functionality as discussed in section 2.2.1.2 on interconnect power dissipation.

For calculation of net power without considering the switching activity, only net length will be used as the criteria for dividing the total interconnect power among different nets as proposed in [20]. We calculate the net power by the formula

$$NP_j = \frac{NL_j}{WL} \times TIP \tag{3.1}$$

where NP_j is the net power, NL_j is the net length of net j in the netlist containing N number of nets, TIP is the total interconnect power and WL is the total wirelength given by

$$WL = \sum_{j=1}^{N} NL_j \tag{3.2}$$

When switching activity is considered in addition to net length, we calculate the net power by the formula

$$NP_j = \frac{(NL_j \times \alpha_j)}{(\sum_{j=1}^{N} (NL_j \times \alpha_j))} \times TIP \tag{3.3}$$

where α_j is the switching activity of net j.

Thus, the amount of net power that contributes to the connecting module b_i of the net j can be stated as follows:

$$MP_{ij} = \frac{A_i}{TA_j} \times NP_j \tag{3.4}$$

where MP_{ij} indicates the amount of power from net j contributing to module b_i, A_i represents the area of functional module b_i and TA_j tells the total area of connected modules of net *j*.

Following the above procedure, we can obtain for a particular module, the contributions of power from each net in the netlist. Besides this, each module has its own internal power consumption stated as the original module power *P* which is obtained experimentally for real circuits. Finally, we sum up such contributions from each net and the original module power to obtain the total power TP_i of each module b_i as

$$TP_i = P_i + \sum_{j=1}^{N} MP_{ij} \qquad (3.5)$$

Figure 3.3 presents the Calculate_Power algorithm for calculating the interconnect aware power values for all the modules. The algorithm has a runtime complexity of $O(nm)$ where *n* is the number of nets in the netlist and *m* is the number of modules in the benchmark.

```
Begin
  For each module $b_i$
        For each net j Є N
              $NP_j = ((NL_j \times \alpha_j) / \sum_{j=1}^{N} (NL_j \times \alpha_j)) * TIP$
              $MP_{ij} = (A_i / TA_j) \times NP_j$
              Calculate $\sum_{j=1}^{N} MP_{ij}$
        End For
  $TP_i = P_i + \sum_{j=1}^{N} MP_{ij}$
  End For
End.
```

Figure 3.3: Pseudo code of the Calculate_Power algorithm.

We now illustrate our algorithm in determining the impact of switching activity of interconnects on power distribution profile with the help of an example. Let there be four blocks A, B, C and D, each of equal dimensions, and with original power P_i values as 5, 1, 1 and 3 units respectively. Let there be two interconnects AB and CD of lengths 2 and 3 units respectively, and having switching activity values of 0.4 and 0.2 respectively. Let the total interconnect power be 4 units. Now, we determine the power density profile for both the cases when switching activity is considered in power estimations and when it is not considered.

Case 1: Power estimations considering only interconnect lengths.

Step 1. Net power $NP_{AB} = \frac{2}{(2+3)} \times 4 = 1.6$

Net power $NP_{CD} = \frac{3}{(2+3)} \times 4 = 2.4$

Step 2. The amount of net power contributing to the connecting modules A and B of the net AB, $MP_A = 1/2 \times 2.9 = 1.45$

and $MP_B = 1/2 \times 2.9 = 1.45$

Similarly, amount of net power contributing to the connecting modules C and D of the net CD, $MP_C = 1/2 \times 1.1 = 0.55$

and $MP_D = 1/2 \times 1.1 = 0.55$

Thus, the total power of each module can be calculated by summing up the original module power with net power contributions to each module. The total power of each module and the power distribution profile is shown in Figure 3.4 below.

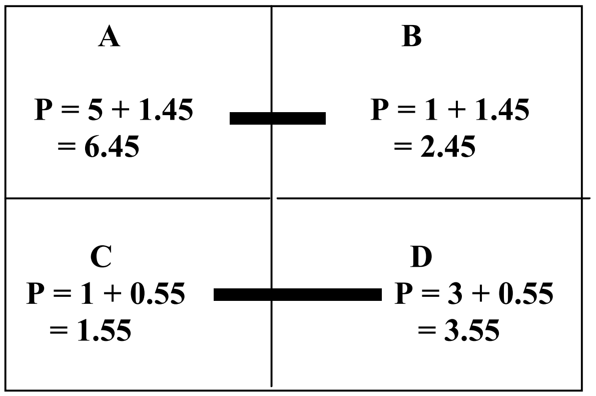


Figure 3.4: Power distribution profile for case 1.

Case 2: Power estimations considering both interconnect lengths and switching activity values.

Step 1. Net power $NP_{AB} = \frac{2 \times 0.4}{(2 \times 0.4 + 3 \times 0.1)} \times 4 = 2.9$

Net power $NP_{CD} = \frac{3 \times 0.1}{(2 \times 0.4 + 3 \times 0.1)} \times 4 = 1.1$

Step 2. The amount of net power contributing to the connecting modules A and B of the net AB, $MP_A = 1/2 \times 1.6 = 0.8$

and $MP_B = 1/2 \times 1.6 = 0.8$

Similarly, amount of net power contributing to the connecting modules C and D of the net CD, $MP_C = 1/2 \times 2.4 = 1.2$

and $MP_D = 1/2 \times 2.4 = 1.2$

Thus, the total power of each module can be calculated by summing up the original module power with net power contributions to each module. The power distribution profile is shown in Figure 3.5 below.

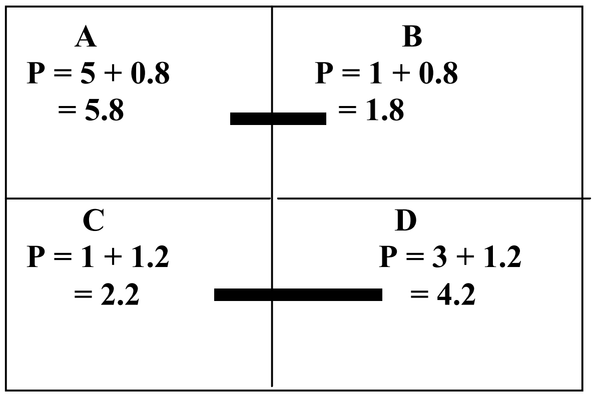


Figure 3.5: Power distribution profile for case 2.

From the above example, we observe different power distribution profiles for the two cases. This will lead to different heat diffusion flows as explained in section 2.2.2 and hence different values of average and peak temperature of the chip. Therefore, switching activity must be considered together with the net length for correct chip temperature estimations.

3.3.3 Temperature Approximation

Since thermal effects are influenced by relative placement of blocks as discussed in section 2.2.2, therefore, it is imperative to include temperature in the cost function in the floorplanning algorithm to achieve an optimum floorplan with reduced hotspots. However, it is prohibitively time consuming to involve the temperature calculations every time when evaluating a large number of solutions during simulation procedure. Other than using the actual temperature values, we have adopted the power density metric as a thermal-conscious mechanism in our floorplanner. We can substitute the temperature for the power density, according to equation (2.15), to approximate the 3-tie temperature function

$$C_T = (T - T_o) / T_o$$

proposed in [17] to reflect the thermal effect on a chip. As such, the 3-tie power density function is defined as

$$P = (P_{max} - P_{avg}) / P_{avg} \qquad (3.6)$$

where P_{max} is the module with the maximum power density while P_{avg} is the average power density of all modules.

The cost function used in simulated annealing in the interconnect and thermal aware floorplanning algorithm can now be written as

$$Cost = \alpha \times A + \beta \times WL + \gamma \times P \qquad (3.7)$$

where α, β and γ are constants which denote the relative weights of A, WL and P

respectively in the cost function, $(\alpha + \beta + \gamma) \leq 1$. At the end of execution of our floorplanner, we obtain the floorplan description and the power numbers for each module including the interconnect power contributions from all nets. We then use the block model of HotSpot to provide the temperature estimations of the chip.

3.4 Experimental Results

To evaluate our interconnect power and thermal aware floorplanning algorithm, we performed a series of three experiments. First experiment was conducted using the traditional area and wirelength metrics in the cost function. It presents peak and average temperatures when only the net length of interconnects is used in deriving the power density profile of the chip. Second experiment includes the effect of switching activity in addition to net length of interconnects in deriving the power density profile of the chip. Finally, we prove the effectiveness of our algorithm in reducing the hotspots in the third experiment.

3.4.1 Experimental Setup

The experimental setup is as follows. The simulated annealing floorplanning algorithm is implemented in C++ programming language on an Intel Pentium 4, 1.73 GHz PC with 1 GB RAM. The operating system is RedHat Linux v6.1, kernel version 2.4. The experiments were performed on a set of five MCNC benchmark circuits that consists of hard modules. We tested all these benchmarks. Table 3.1 gives the information of MCNC benchmarks.

Table 3.1. MCNC benchmarks information.

Circuit	Block #	Net #	Pin #	Pad #
apte	9	97	214	73
xerox	10	203	696	2
hp	11	83	264	45
ami33	33	123	480	42
ami49	49	408	931	22

The widely used method of half-perimeter bounding box (HPWL) model as explained in section 2.1.2 is adopted to estimate the wire length. Power values in the range of 0.05mW to 3W were randomly assigned to the modules in different benchmarks due to lack of information on the internals of each module. These power values are the typical values in modern high performance circuits such as microprocessors. The total net power is assumed to be 30% of total power of modules due to lack of information for the MCNC benchmarks. Random values in the range 0-1 were assigned to switching activity for various interconnects as we do not know the internal functionality of the MCNC benchmark circuits. Random selection of values for module power and switching activity of various interconnects will impact the runtime complexity of the algorithm further by $O(n+m)$ where n is the number of nets in the netlist and m is the number of modules in the benchmark.

In the simulated annealing process, the temperature was decreased at a constant rate (0.9). We terminate the annealing process if the rejection rate of moves exceeds the convergent rate of 0.85 at a certain temperature or the temperature decreases beyond the threshold value of 0.1.

The final results are based on average of the results of 100 test runs for each benchmark circuit for a particular set of power values of the modules. We consider area optimization as the main criterion in the cost function to obtain floorplans with the smallest area possible for all the benchmarks.

3.4.2 Results and Analyses

Table 3.2 shows the experiment results of our approach when considering only the traditional metrics (area and wire) with and without the consideration of switching activity of interconnects. We list the area, wirelength, T_{peak}, T_{avg} and deadspace for each of the circuit. The run time of the approach is also provided. Note that the area, wirelength, deadspace and runtime for both the experiments are the same. Only values of T_{peak} and T_{avg} are affected by the inclusion of switching activity of interconnects.

Table 3.2. Results considering traditional area and wirelength minimizations.

Circuit	Area (mm^2)	WL (mm)	T_{peak} (^oC)	T_{avg} (^oC)	T_{peak} (Sa) (^oC)	T_{avg} (Sa) (^oC)	Run Time (Sec)	Dead Space (%)
apte	47.31	653.51	90.15	62.63	91.25	63.87	2.48	1.59
xerox	20.42	402.88	156.55	85.08	169.75	99.90	4.47	5.26
hp	9.20	280.79	396.75	164.52	410.75	174.37	5.47	4.03
ami33	1.22	83.74	964.67	810.33	979.94	818.34	53.68	5.40
ami49	38.84	1061.42	599.87	294.14	605.75	298.42	60.13	8.75

The peak and average temperature results from these tables also reiterate the importance of including the switching activity of interconnects in determination of power consumption, which most prior works ignore. The difference between peak and average temperatures not considering switching activity of interconnects in power estimation (T_{peak} and T_{avg}) and considering it (T_{peak} (Sa) and T_{avg} (Sa)) is 15^oC on the average.

Table 3.3 presents the results of applying our interconnect power and thermal aware floorplanning algorithm. When taking thermal effect into account together with the area and wirelength metrics, our floorplanner can reduce the peak and average temperatures by as much as 20% while increasing the wirelength by 2% and providing a comparable chip area as compared to the floorplan generated using traditional metrics.

Table 3.3. Results using our thermal aware floorplanner.

Circuit	Area (mm^2)	WL (mm)	T_{peak} (Sa) (^oC)	T_{avg} (Sa) (^oC)	Run Time (Sec)	Dead Space (%)
apte	47.31	653.51	89.85	62.59	3.06	1.59
xerox	20.42	410.26	125.34	80.65	5.49	5.26
hp	9.33	283.83	316.25	162.20	6.87	5.37
ami33	1.22	86.55	907.45	803.56	65.70	5.40
ami49	39.05	997.65	531.95	284.77	71.12	9.24

Figures 3.6 and 3.7 show the peak and average temperatures respectively of all five MCNC benchmarks with/without consideration of switching activity of interconnects for area optimization only as well as area and thermal optimizations. It shows that the area optimization with the consideration of switching activity (Sa) of interconnects in deriving power dissipation results in highest peak and average temperatures for all benchmarks. The temperature can be effectively reduced through interconnect power and thermal aware optimization combined with area constraint such that lower temperature is achieved with the same compact floorplan. The hotspot temperature (peak temperature) is reduced by as much as 20% in circuit *xerox* in Figure 3.6. The overall average temperature is reduced by around 1~19 ^oC as shown in Figure 3.7.

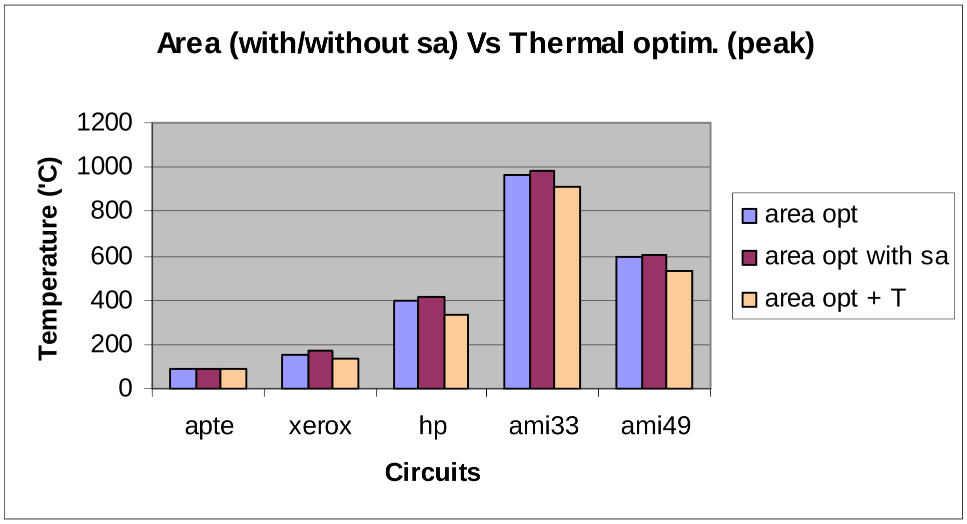


Figure 3.6: Comparison of peak temperatures for area optimization with/without consideration of switching activity and thermal aware optimization.

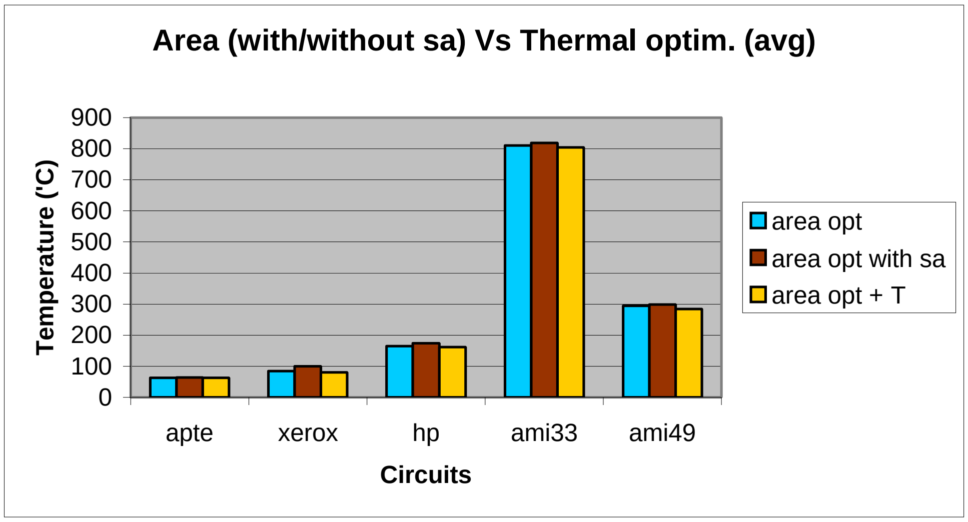


Figure 3.7: Comparison of average temperatures for area optimization with/without consideration of switching activity and thermal aware optimization.

However, some benchmarks could not achieve large temperature reduction. For example, the hotspot temperature reduction for *apte* is only 1.4^{o}C. The reason behind this is that the power density is relatively high for a limited small area and deadspace is very less to allow heat in hotter regions to flow through more dead areas that consume no power. Thus, there is not much flexibility for algorithm to discover a good solution. As observed from Table 3.3, *apte* has a very small deadspace of 1.59%, thus restricting the heat flow and hence the temperature reduction. *xerox* has a fairly large deadspace of 5.26% and relatively larger area compared to other benchmark circuits, which allow greater heat flow and hence it achieves large temperature reduction by 20%.

Note that the runtime is almost comparable in both the cases i.e. using traditional metrics and our thermal aware approach.

Table 3.4. Comparison of runtime results for Hung's floorplanner and our floorplanner.

Circuit	**Hung's floorplanner Dual Intel Xeon (3.2 GHz, 2GB RAM) Run Time (Sec)**	**Our floorplanner Intel Pentium4, (1.73 GHz, 1 GB RAM) Run Time (Sec)**
apte	-	3.06
xerox	13	5.49
hp	25.78	6.87
ami33	101	65.70
ami49	240	71.12

Table 3.4 shows the runtime comparisons between W.L. Hung's 2D floorplanner [20] and our floorplanner using the MCNC benchmarks. We note that both the floorplanners use the randomly selected values in the similar ranges for the power of different blocks in the benchmark circuits and the total net power, thus affecting the run time complexity in a similar way. However, our floorplanner uses additional randomly selected values for the switching activity of different nets, which can slightly increase the run time. But as observed from the table above, clearly, our floorplanner has a much shorter runtime, thus proving the simplicity of our interconnect power and thermal aware

floorplanning algorithm. Other parameters like area and wirelength are not considered for comparison because we have used hard modules in our work where as W.L Hung's floorplanner is based on soft modules, which achieves much larger reduction in area due to adjustment of aspect-ratio of the soft blocks. Since wirelength estimates depend on the relative positions of different blocks and eventually their area, therefore, they are also not considered for comparison.

To show the pre-thermal and post thermal effects on the arrangement of blocks, we present the floorplan layout for all the benchmarks for both the above cases. Figures 3.8 to 3.16 show the floorplans of various benchmarks using the area optimization factor only and that using our interconnect power and thermal aware floorplanning algorithm.

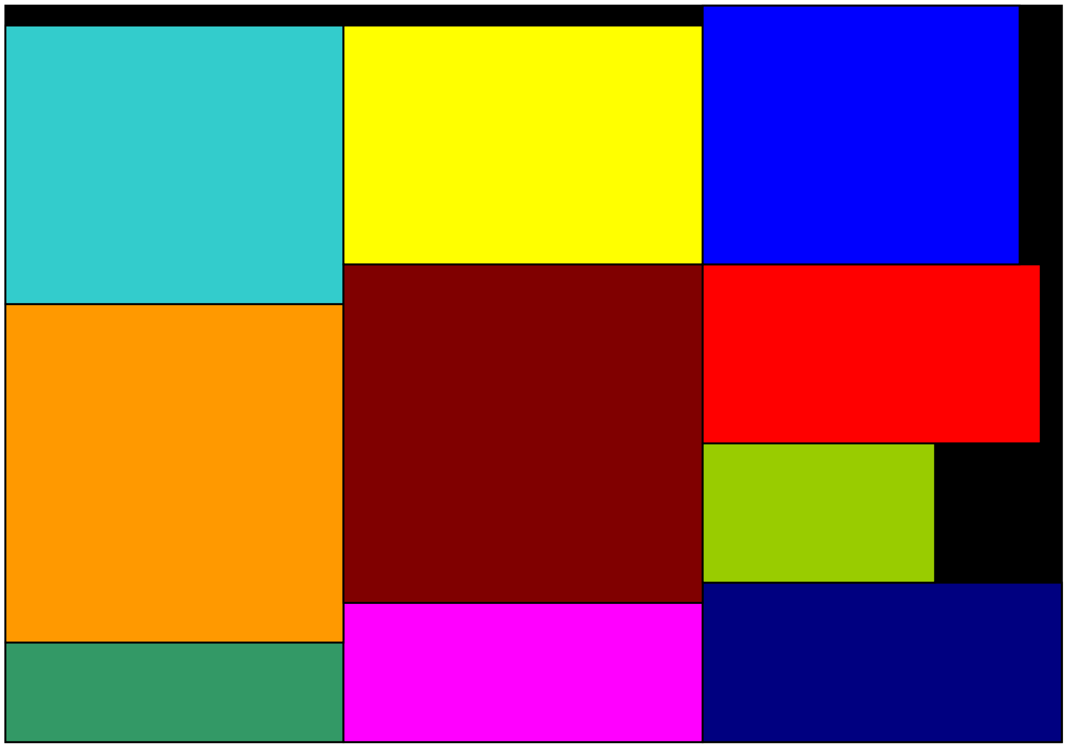


Figure 3.8: Floorplan of *xerox* with area optimization only.

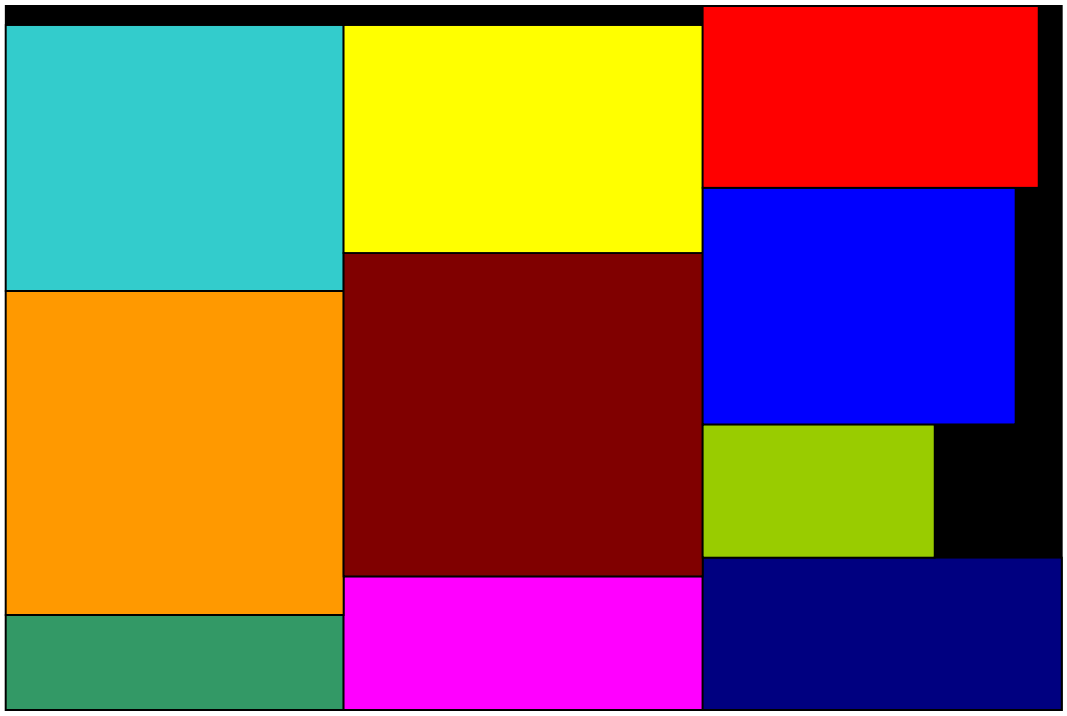


Figure 3.9: Floorplan of *xerox* with area and thermal optimization.

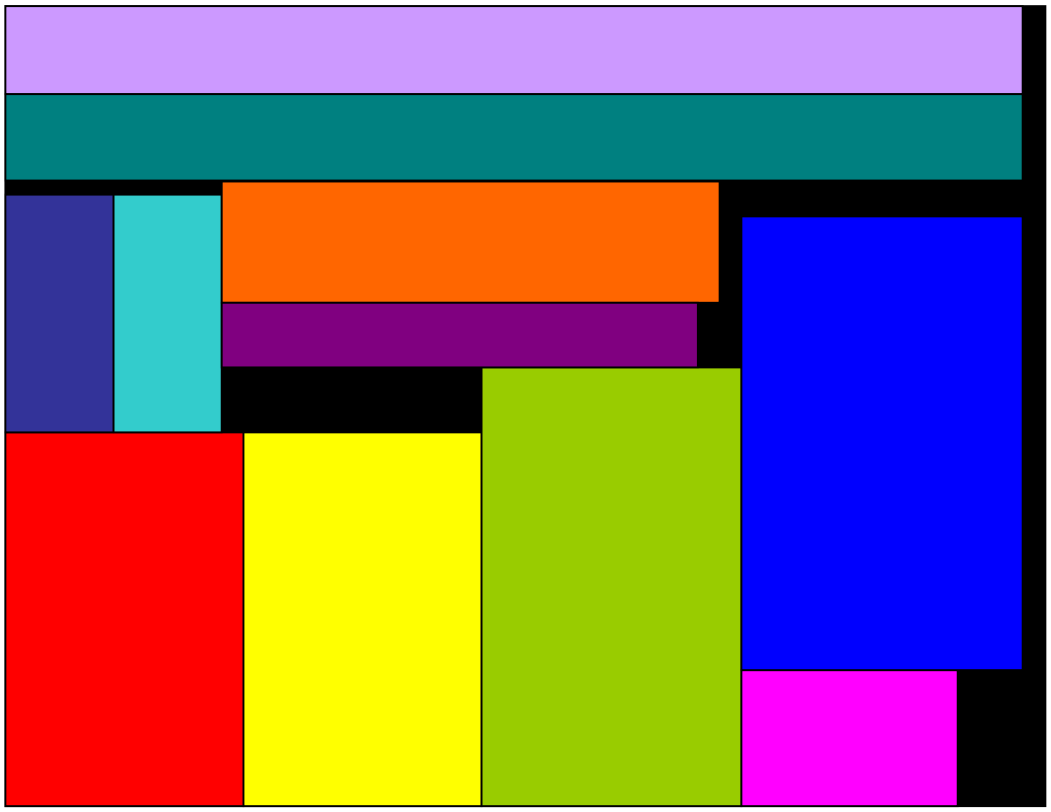


Figure 3.10: Floorplan of *hp* with area optimization only.

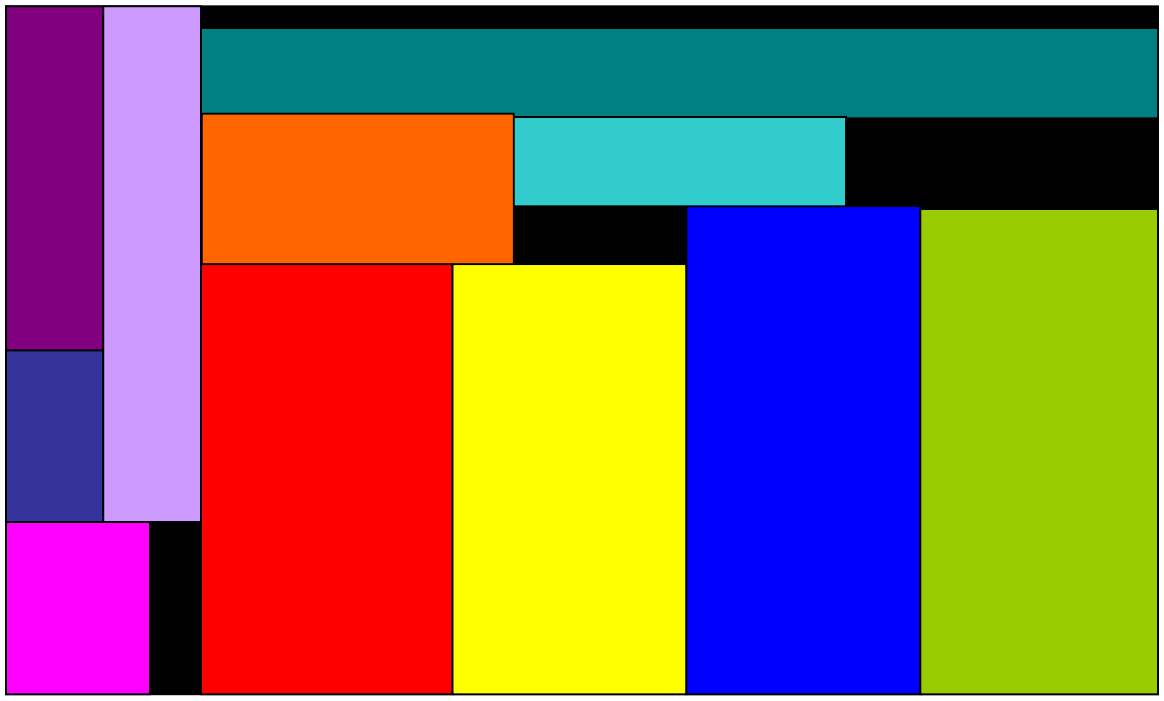


Figure 3.11: Floorplan of *hp* with area and thermal optimization.

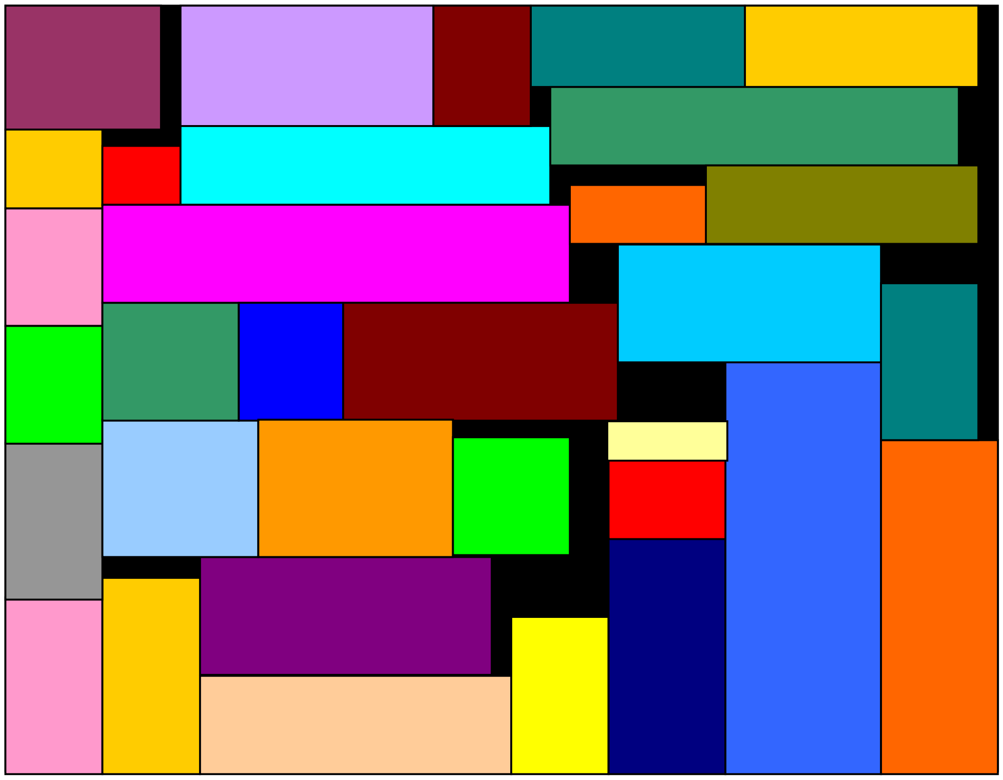


Figure 3.12: Floorplan of *ami33* with area optimization only.

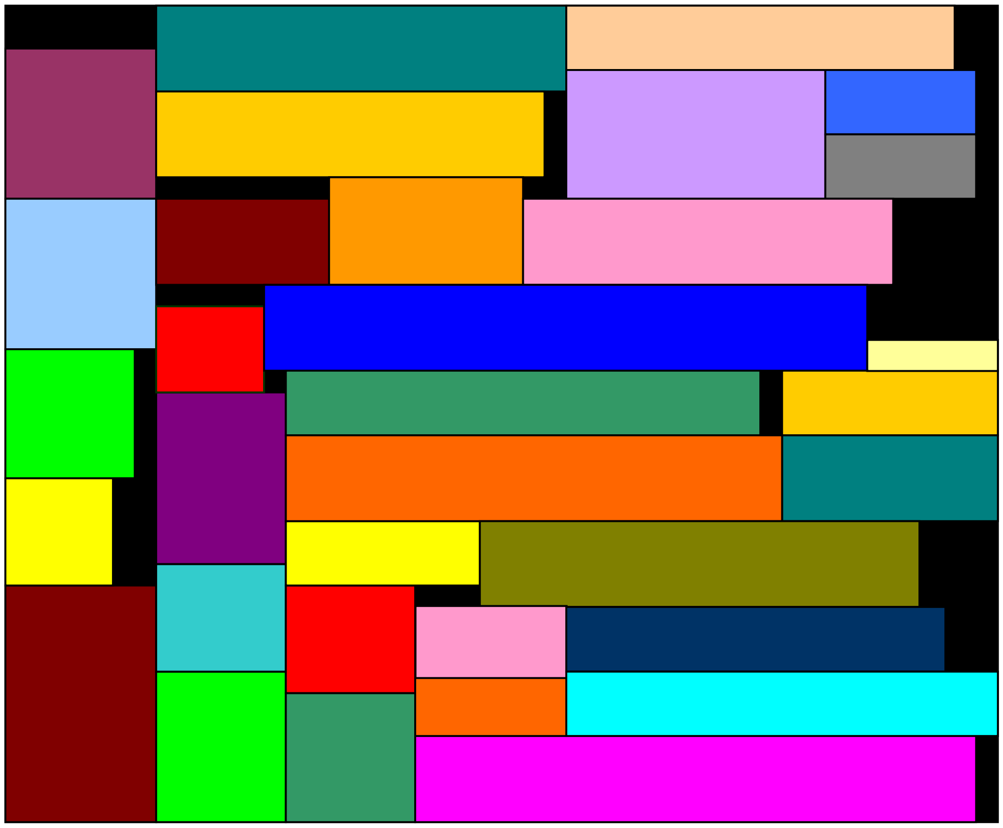


Figure 3.13: Floorplan of *ami33* with area and thermal optimization.

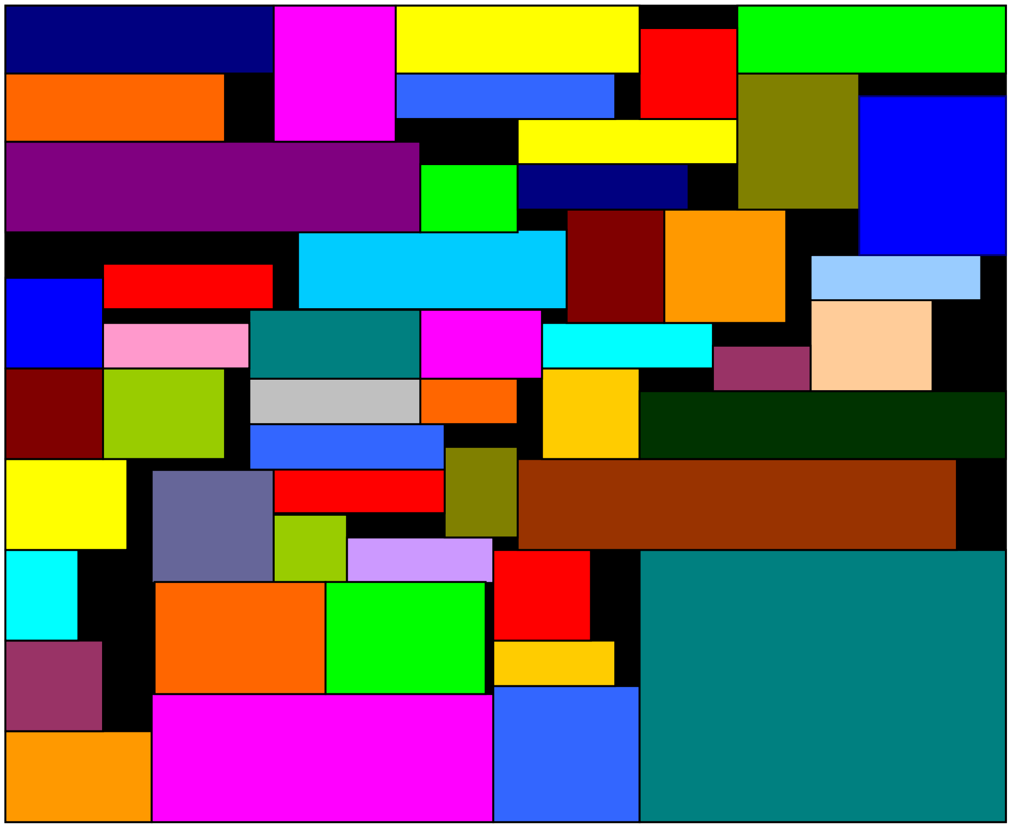


Figure 3.14: Floorplan of *ami49* with area optimization only.

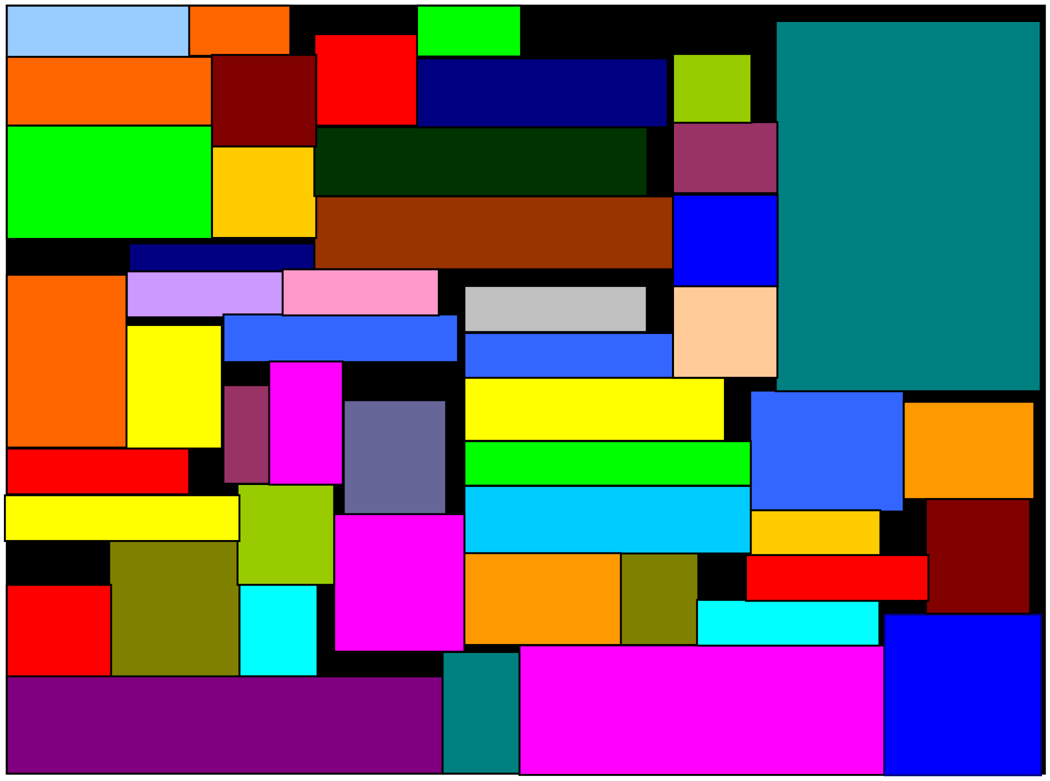


Figure 3.15: Floorplan of *ami49* with area and thermal optimization.

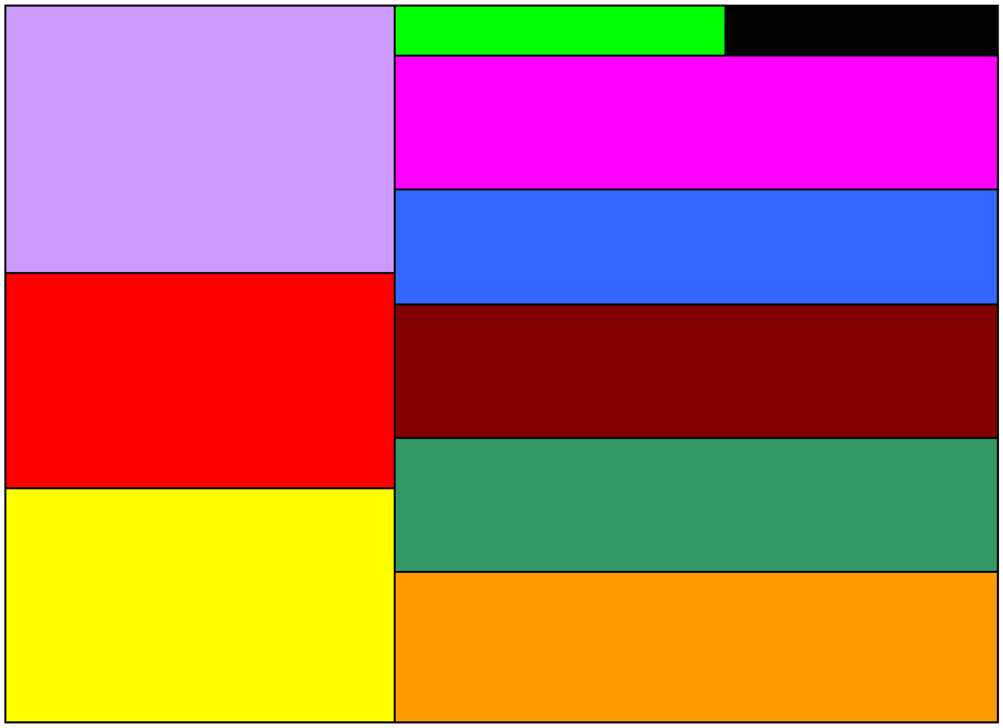


Figure 3.16: Floorplan of *apte* with area optimization only, and area and thermal optimization.

From the above figures, we observe that the floorplan layout changes considerably after the thermal optimization for most of the benchmarks. However, for the benchmark *apte*, we observe that the floorplan layout remains the same before and after the thermal optimization though the temperature results are different for both the cases as seen from the Tables 3.2 and 3.3. Also, the wirelength value changes. This can be explained by considering the fact that one or many of the blocks would have undergone double rotations at the same location in order to meet the thermal constraints as specified in the cost function. This results in the change of pin locations and hence the netlength. Thus, the power distribution contribution from the nets which have their netlength value changed will affect the power density profile of the chip. Hence, we get different temperature estimations, compared to traditional area optimization, in the thermal optimization approach.

3.4.3 Discussion

Since our goal is to mainly study the impact of power distribution profile of the chip, obtained by including the power dissipation, considering switching activity of the interconnects, on chip temperature estimations, we have presented the results for the experiments performed with only one particular set of power values randomly chosen for all the modules in the benchmark. However, we also performed experiments with different set of power values and observed a similar behavior in the results for temperature estimations. Thus, we can safely draw conclusions about the variations in temperature estimations with and without considering the switching activity of interconnects, and our interconnect power and thermal aware floorplanning algorithm from only one set of experimental results.

However, since the power distribution profile depends on both the netlength and switching activity of interconnects, we chose to run the tests 100 times to ensure that we cover the maximum variation in temperature estimations that can be evaluated by assigning different sets of random values to switching activity of each interconnect.

As the cost function plays an important role in selecting the quality of the floorplan based on some specific criteria, we arbitrarily assigned the values of α, β and γ in the cost function and perform the entire set of experiments with those fixed values to obtain the same quality of floorplans for different benchmarks. This avoids unambiguous interpretation of results.

We can achieve further temperature reductions by using our interconnect power and thermal aware floorplanning algorithm if we relax the area and wirelength constraints. This will allow more dead areas to be created to allow heat in hotter regions to flow through them that consume no power. But this is beyond the scope of our goal of interconnect power and thermal aware floorplanning problem stated in section 3.2.

Based on the above results and discussion, we state that more attention should be drawn to switching activity of interconnects in future technologies and it is imperative to include switching activity together with length of interconnects in deriving power estimates in guiding any thermal-aware floorplanning.

3.5 Conclusion

In this chapter, we have presented our interconnect power and thermal aware floorplanner based on B*-Tree representation. The problem has been formulated as a floorplanning optimization problem under interconnect power dissipation considerations. Then, we have explained the entire methodology followed in developing the algorithm. We have discussed how to incorporate switching activity of interconnects in determining power dissipation estimations. Further, we have shown how to make temperature approximations, and the effect of placement and power density profile on temperature estimations. The experimental results prove the effectiveness of our algorithm in reducing hotspots and show that it performs better than W.L. Hung’s 2D floorplanning algorithm [20] in terms of run time when testing on a set of MCNC benchmark circuits.

Chapter 4

Variability Aware Floorplanning

The scaling of technologies towards the nanometer regime brings with it a challenging increase in the amount of variability due to uncertainty in initial estimates in the early phases of chip design. As the block and interconnect parameters such as block dimensions show variability due to design uncertainty, the prediction of circuit performance is becoming a challenging task in early chip planning stage. To address the variability issue at the floorplanning stage, we develop a variability-aware floorplanner based on analytical approach to determine the best relative floorplan for all blocks with variable block characteristics. It can predict the ranges of area and wirelength of a design. This early prediction helps estimate the variability impacts on performance parameters such as delay at higher abstraction.

The remainder of the chapter is organized as follows. Section 4.1 describes the related work that has served as a source of motivation for this work. Section 4.2 gives the problem definition. Section 4.3 presents the variability-aware floorplanning algorithm based on affine arithmetic. Section 4.4 presents our experimental results and analyses. Finally, we conclude this chapter in the last section.

4.1 Motivation

In this section, we discuss some related work and the motivation for the research work presented in this chapter. Previous relevant works fall into two broad categories- the first is the prevalence of floorplanner of uncertain designs and the second is the various analytical approaches available for modeling variations.

The first category of work studied floorplanning of uncertain designs. Bazargan [2] developed a Nostradamus floorplanner for slicing floorplans to handle variations in block dimensions, introduced due to design uncertainty, at an early chip planning stage. He established that traditional floorplanners are incapable of handling uncertainty. Bazargan considered the variations in block dimensions by creating distribution lists, each of which consisted of a pair of numbers: width/height of a block and its probability. Only certain discrete values and combinations of width and height for each block were accounted for. However, in practical problems where the internals of the design block are still not clear in the early decision stage, block dimensions can assume any value in a specified range of values. Our work takes into account the entire range of values for the block dimensions and all possible combinations of width and height for each block. Moreover, we develop the floorplanner for non-slicing floorplans and consider the wirelength estimations besides the chip area when testing on MCNC benchmarks.

The second category of related work dealing with the variability effects [32]-[35] falls in to the area of statistical static timing analysis (STA) that uses analytical approaches to find closed-form expressions for the distributions of the circuit delay under the presence of process variations. These methods use normal distributions [30], [31], interval valued analysis, probabilistic intervals [40] or mathematical statistical models for predicting the circuit performance parameters affected by variability. Our motivation of adopting the approach of analyzing the impact of variations in dimensions of modules on the floorplan metrics like chip area and wirelength stem from the work done in the area of timing analysis.

Since floorplanning is an important stage in the VLSI design process that can impact many design decisions, we intend to study the effects of variability on floorplan metrics and modify our floorplanning algorithm accordingly to make it less susceptible to variation effects due to uncertainty in initial estimates in the design prototyping stage.

4.2 Problem Formulation

We define our problem in this work as follows: given the information of a set of modules including their areas and interconnections, the variability-aware floorplanning problem is that of placing the modules in the chip area satisfying a set of conditions and achieving the goal of determining the best relative floorplan with smallest range and average values of area and wirelength for specified variations in dimensions for each block while optimizing area and wirelength. This relative floorplan will ensure that it is least impacted by the effects of design uncertainty at an early stage, and thus, the resulting circuit performance parameters will be effected only slightly. In this work, we consider only the hard modules, i.e. modules that are not flexible in shape but are free to move and rotate.

Let $B = \{b_1, b_2, \ldots, b_n\}$ be a set of n rectangular modules with block b_i of width w_i and height h_i, $1 \leq i \leq n$, such that w_i lies in the range $[R_{mini}, R_{maxi}]$ and h_i lies in the range $[S_{mini}, S_{maxi}]$ where R_{mini}, R_{maxi} and S_{mini}, S_{maxi} are the minimum and maximum values for w_i and h_i respectively. Figure 4.1 illustrates the ranges for width and height of a module. Each module is free to rotate.

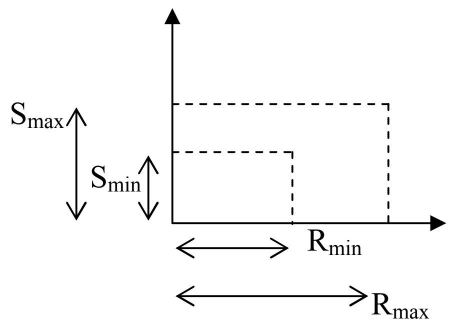


Figure 4.1: Width and height ranges of a module.

Let (x_i, y_i) denote the co-ordinates of the bottom-left corner of the rectangle b_i, on a chip. A floorplan F is an assignment of x_i and y_i that lie in the ranges $[x_{mini}, x_{maxi}]$ and $[y_{mini}, y_{maxi}]$ respectively for each b_i such that no two modules overlap. Figure 4.2 illustrates the possible location of left bottom coordinates (x, y) of the module within the bounding rectangle.

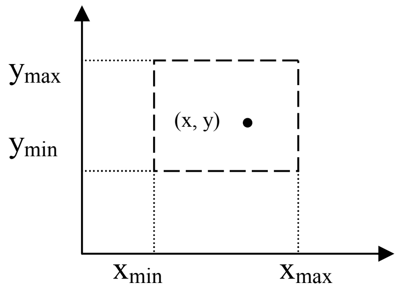


Figure 4.2: Left bottom co-ordinate ranges (x, y) of a module.

The goal of floorplanning algorithm is to minimize

(i) Chip area (i.e. minimum bounding rectangle of F),

(ii) Total wirelength (i.e. the summation of half bounding box of interconnections) induced by the assignment of b_i's and

(iii) The range for total area $[A_{min}, A_{max}]$ and wirelength $[WL_{min}, WL_{max}]$ obtained where A_{min}, A_{max} and WL_{min}, WL_{max} are the lowest minimum and maximum values for total area and wirelength respectively, that can be computed for the placement of B (i.e. a set of n rectangular modules).

Since we can have several relative floorplans F_i in the solution space, we determine the best relative floorplan with the help of the cost function. The criteria listed above for determining the best solution, i.e. the goal of floorplanning problem can be mathematically formulated as a cost function given by the equation

$$C_i = \alpha \times A_i + \beta \times WL_i \tag{4.1}$$

where A_i and WL_i are total area and wirelength metrics respectively for relative floorplan F_i, and α and β are constants which denote the relative weights of total area and wirelength respectively in the cost function, $(\alpha + \beta) \leq 1$. The objective of floorplanning is to find the floorplan with minimum value of C_i i.e.

$$\textit{Best floorplan} \Rightarrow \min\,(C_1, C_2, \ldots, C_m) \tag{4.2}$$

where m is the total number of relative floorplans evaluated. We will determine A_i and WL_i later in section 4.3.

We illustrate the impact of variations in dimensions of each module on the range of chip area by an example. Let us consider a set $B = \{b_1, b_2, b_3, b_4, b_5\}$ of five rectangular modules with each block b_i having variable width w_i and height h_i. Ranges of width and height for each block are listed in Table 4.1. Let Figure 4.3 (a) show the floorplan layout of the chip for a set of worst case randomly chosen values for width and height of each module. If we vary the values for width and height of each module such that the relative positions of modules remain the same, we get a different floorplan layout as shown in Figure 4.3 (b). Values of width and height for each module and their locations in (x, y) co-ordinates are listed in Table 4.2 for both the cases.

Table 4.1. Ranges of width and height for each block.

Block No.	W_i range $[R_{min}, R_{max}]$	H_i Range $[S_{min}, S_{max}]$
1	[1,5]	[2,4]
2	[1,3]	[2,4]
3	[1,4]	[2,6]
4	[1,2]	[2,5]
5	[2,4]	[2,4]

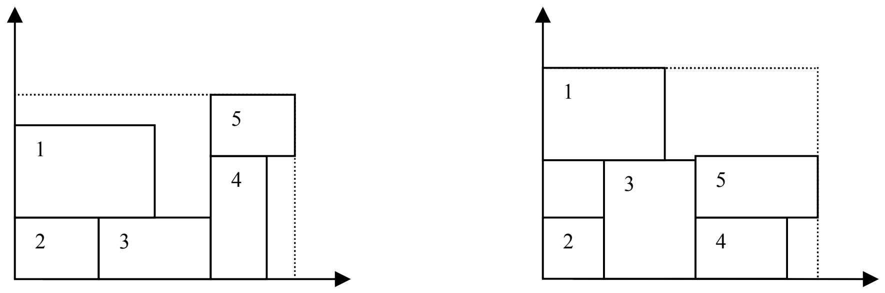


Figure 4.3: (a) and (b): Impact of variability on chip area and location of modules.

Table 4.2. Values of width, height and location of each block.

	Case 1: Figure 4.3 (a)		Case 2: Figure 4.3 (b)	
Block No.	(w, h) for each block	Position (x, y)	(w, h) for each block	Position (x, y)
1	(4, 4)	(0, 3)	(3.5, 4)	(0, 5)
2	(2, 2.5)	(0, 0)	(1.5, 2.5)	(0, 0)
3	(3, 2.5)	(2, 0)	(2.5, 5)	(1, 0)
4	(1, 5)	(5, 0)	(2, 3)	(3, 0)
5	(2, 2.5)	(3, 5)	(3, 2.5)	(3, 3)

On comparing Figures 4.3 (a) and (b), we can see that both have different values for width, height and location of left bottom co-ordinates (*x*, *y*) for each module as tabulated in Table 4.2. Table 4.3 presents the width, height and area (bounded rectangle) of the floorplan layout in both the cases.

Table 4.3. Width, height and area of the floorplan layout of Figures 4.3 (a) and (b).

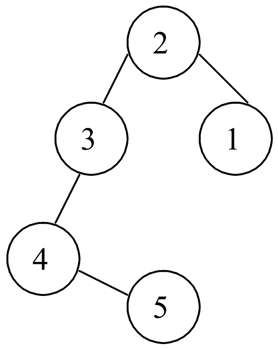
Floorplan Metrics	**Case 1: Figure 4.3 (a)**	**Case 2: Figure 4.3 (b)**
Width	6	6
Height	10	9
Area	60	54

From Table 4.3, we observe that the value of chip area changes on changing the dimensions of each module. By trying all the possible combinations of different values of dimensions for each module, we get different floorplan layouts with different values of chip area. Thus, variability in dimensions produces variations in chip area. We intend to determine the range of variations in area and wirelength for a given set of modules, each with specified range of variations in dimensions, in our solution approach.

4.3 Algorithm

In this section, we describe the methodology adopted to solve the problem outlined in section 4.2. We first develop the simulated annealing floorplanning algorithm based on Monte Carlo simulation (MC) technique which is widely used to solve complex numerical problems. We then develop our own algorithm based on affine arithmetic to determine the ranges of chip area and wirelength. We use B*-Tree representation for our floorplanning algorithm as it is easy to implement, has a smaller solution space and time complexity, and many other advantages as outlined in chapter 2.1.1.4.

We first illustrate our solution approach by a simple example discussed previously in section 4.3. Figure 4.4 (a) shows the B*-Tree corresponding to the floorplan in Figure 4.3. Figure 4.4 (b) shows the B*-Tree obtained by perturbing the B*-Tree in Figure 4.4 (a).



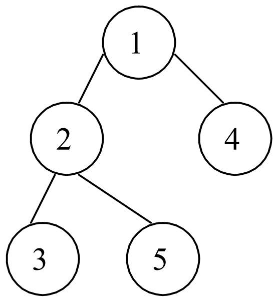
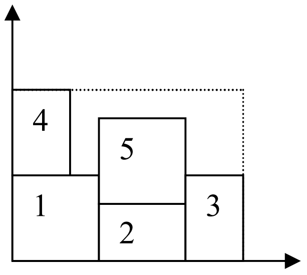


Figure 4.4: (a) B*-Tree 1 (b) B*-Tree 2.

Now, if we repeat the same steps as followed in example in Figure 4.3, we obtain two sets of floorplans as shown in Figures 4.5 (a) and (b) with worst case random values of width w_i and height h_i for each block b_i. Values of width and height for each module and their locations in (*x, y*) co-ordinates are listed in Table 4.4 for both the cases.



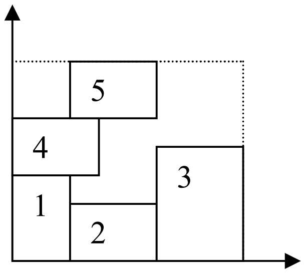


Figure 4.5: (a) and (b): Impact of variability on chip area and location of modules for B*-Tree 2.

Table 4.4. Width, height and location of each block for B*-Tree 2.

Block Information			**Case 1: Figure 4.5 (a)**		**Case 2: Figure 4.5 (b)**	
Block No.	**W_i range [R_{min}, R_{max}]**	**H_i Range [S_{min}, S_{max}]**	**(w, h) for each block**	**Position (x, y)**	**(w, h) for each block**	**Position (x, y)**
1	[1, 5]	[2, 4]	(2, 4)	(0, 0)	(1, 4)	(0, 0)
2	[1, 3]	[2, 4]	(2, 3)	(2, 0)	(2, 3)	(1 ,0)
3	[1, 4]	[2, 6]	(1, 4)	(4, 0)	(2, 6)	(3, 0)
4	[1, 2]	[2, 5]	(1.5, 4)	(0, 4)	(2, 3)	(0, 4)
5	[2, 4]	[2, 4]	(2, 4)	(2, 3)	(2, 3),	(1, 7)

Table 4.5 presents the width, height and area (bounded rectangle) of the floorplan layout in both the cases.

Table 4.5. Width, height and area of the floorplan layout of Figures 4.5 (a) and (b).

Floorplan Metrics	Case 1: Figure 4.5 (a)	Case 2: Figure 4.5 (b)
Width	6	5
Height	9	10
Area	54	50

On comparing the areas of Figures 4.3 (a) and (b) with those of Figures 4.5 (a) and (b) from Tables 4.3 and 4.5, we can see that the B*-Tree in Figure 4.4 (b) gives a smaller value of maximum area and range as compared to that given by B*-Tree in Figure 4.4 (a). Since the goal of floorplanning is to determine the smallest range and average value of total area, the floorplans corresponding to B*-Tree in Figure 4.4 (b) will be considered the best relative floorplan. Similarly, we can prove the optimality for the wirelength range also.

To determine the best relative floorplan corresponding to a unique B*-Tree with smallest range and average values of area and wirelength for specified variations in dimensions for each block, we develop a simulated annealing based algorithm based on B*-Tree as explained in chapter 2.1.2.4 for handling the placement with variations in dimensions of each module. The algorithm perturbs the B*-Tree to another B*-Tree by using the following three operations.

Op1: Rotate a module.

Op2: Move a module to another place.

Op3: Swap two modules.

We apply both MC approach and our approach to perform placement for each feasible B*-Tree to determine the ranges of location of each module, total area [A_{min}, A_{max}] and wirelength [WL_{min}, WL_{max}] parameters. The placement algorithm is discussed in detail in subsequent sections for both the approaches. The perturbation process repeats until pre-defined termination conditions are met. The termination condition checks for the temperature value if it is greater than the threshold and convergent rate of solutions is greater than rejection rate. The best relative floorplan corresponds to the feasible B*-Tree which gives the smallest range and average values for chip area and wirelength. In other words, among all the relative floorplans F_i, best relative floorplan is the one that has the lowest value of cost function. Cost function C_i is modified from equation (4.1) to

$$C_i = \alpha * (A_{mean} + A_{range})_i + \beta * (WL_{mean} + WL_{range})_i \tag{4.3}$$

where α and β are constants and

$$A_{range} = (A_{max} - A_{min}), \text{ and} \tag{4.4}$$

$$A_{mean} = (A_{min} + A_{max}) / 2, \tag{4.5}$$

$$WL_{range} = (WL_{max} - WL_{min}) \text{ and} \tag{4.6}$$

$$WL_{mean} = (WL_{min} + WL_{max}) / 2 \tag{4.7}$$

where A_{min}, A_{max} and WL_{min}, WL_{max} are the lowest minimum and maximum values for total area and wirelength respectively, that can be computed for the placement of B corresponding to a particular B*-Tree. Figure 4.6 summarizes the flow of our algorithm.

```
Input: A set of modules $b_i$ ($w_i$ Є [$R_{mini}$, $R_{maxi}$], $h_i$ Є [$S_{mini}$, $S_{maxi}$]) and Netlist N
Output: Floorplan F of ranges of chip area [$A_{min}$, $A_{max}$] and total wirelength [$WL_{min}$, $WL_{max}$] and containing modules with ranges of locations [$x_{mini}$, $x_{maxi}$] and [$y_{mini}$, $y_{maxi}$] for each $b_i$.
Begin
    Initialize temperature T
    Initialize a B*-Tree for the input blocks
  // Run B*-Tree based simulated annealing floorplanning algorithm.
    Do
      Repeat
        Perturb
        Placement
        Compute Cost
        If Cost < Previous cost
            Accept the move
        Else
            Prob = min (1, $e^{-k\Delta c/T}$), where $\Delta_c$ = change in cost, k=constant
            Rand = Random (0, 1)
            If Rand $\leq$ Prob then
                Accept the move
            Else
                Reject the move
            End if
        End if
      Until (Uphill moves > 2*N or Downhill moves < N)
      T = 0.9 * T
    While (T $\geq$ Threshold and Reject_rate $\leq$ Converge_rate)
End.
```

Figure 4.6: Pseudo code of our floorplanning algorithm.

4.3.1 Monte Carlo Simulation Approach

In the Monte Carlo simulation approach, we randomly generate the values for width and height that lie in the ranges [R_{mini}, R_{maxi}] and [S_{mini}, S_{maxi}] respectively for each block b_i in the benchmark. We then do the placement for the corresponding B*-Tree as outlined in chapter 2.1.1.4. This process is repeated several times ($k \sim 2^n$) to obtain several different floorplans with different combinations of random values of dimensions for each block. The range and average values of area and wirelengths are computed according to equations 4.4, 4.5 and 4.6, 4.7 respectively for these set of floorplans corresponding to the unique feasible B*-Tree. Figure 4.7 presents the pseudo code for placement using MC approach. The complexity of MC approach is $O(n2^n)$ where n is the number of modules.

```
Begin
  Loop: Iterate (k ~ 2^n) times
        For each $b_i$
            Randomly assign ($w_i \in [R_{mini}, R_{maxi}]$ and $h_i \in [S_{mini}, S_{maxi}]$)
        End For
        Calculate ($x_i$, $y_i$) for each $b_i$ according to conventional B*-Tree placement procedure.
        Calculate total Area A and Wirelength WL
  End Loop
        Calculate $A_{mean}$, $A_{range}$, $WL_{mean}$ and $WL_{range}$
End.
```

Figure 4.7: Pseudo Code of placement using MC approach.

4.3.2 Our Approach

In our algorithm approach, we evaluate the ranges of area and wirelength of a relative floorplan corresponding to each feasible B*-Tree by applying a series of affine arithmetic operations on affine operands as discussed below. During placement, we replace interval bound for width $w = [R_{mini}, R_{maxi}]$ and height $h = [S_{mini}, S_{maxi}]$ of each module b_i by an affine form

$$\hat{w} = w_{0i} + w_{ki}\varepsilon_{ki},$$

where $w_{0i} = (R_{mini} + R_{maxi}) / 2$ and

$w_{ki} = (R_{maxi} - R_{mini}) / 2$

$$\text{and } \hat{h} = h_{0i} + h_{ki}\varepsilon_{ki},$$

where $h_{0i} = (S_{mini} + S_{maxi}) / 2$ and

$$h_{ki} = (S_{maxi} - S_{mini}) / 2$$

We assign the co-ordinates (x_i, y_i) for each b_i such that x_i and y_i are interval bound and represented by an affine form.

$$\hat{x} = x_{0i} + x_{ki}\varepsilon_{ki} \text{ and } \hat{y} = y_{0i} + y_{ki}\varepsilon_{ki}$$

We now describe the procedure for obtaining the placement, i.e. the range of location of each module, from a corresponding B*-Tree. We determine the ranges of x and y co-ordinates for each module as discussed below.

4.3.2.1 Determination of *x* Co-ordinates

The root of a B*-Tree corresponds to the module on the bottom-left corner with coordinates (0, 0). The left child n_j of a node n_i denotes the module m_j that is the lowest adjacent module on the right-hand side of m_i, i.e. $x_j = x_i + w_i$, according to equation 2.2. We represent x_j in the affine form as a result of addition operation on two other affine operands x_i and w_i as

$$x_j = x_{0i} + w_{0i} + (x_{ki} + w_{ki})\,\varepsilon_{ki} \quad (4.8)$$

The right child n_k of a node n_i denotes module m_k that is the lowest visible module above m_i and with the same x co-ordinate as m_i, i.e. $x_k = x_i$, according to equation 2.3. x_k is represented in the affine form as

$$x_k = x_{0i} + x_{ki}\,\varepsilon_{ki} \quad (4.9)$$

We thus perform addition and assignment affine operations according to equations (4.8) and (4.9) to determine the x co-ordinate of each module in the affine form depending on whether it is the left or the right child of any module in the B*-Tree. We then convert the affine expression for x co-ordinate to IA form to yield an almost-exact range [x_{mini},

x_{maxi}] for each b_i.

4.3.2.2 Determination of *y* Co-ordinates

We use the contour structure as explained in chapter 2.1.1.4 to find the y co-ordinate of a module. Recall, that we first define a permutation π which is the label sequence when we traverse the tree in depth-first search order. The first element in permutation π is the root of the tree. The contour structure is a doubly linked list of modules, which describes the contour line in the current compaction direction. For each module m_i, let $\psi(i)$ be the set of modules m_k with its order lower than m_i in permutation π and interval $(x_k, x_k + w_k)$ overlaps interval $(x_i, x_i + w_i)$ by a non-zero length. Since each variable is represented in the affine form, we define the absolute intervals for each variable as computed below.

- x_k lies in the range $[x_{mink}, x_{maxk}]$ and w_k in the range $[R_{mink}, R_{maxk}]$.
- $x_k + w_k$ gives an interval $[(x_{mink} + R_{mink}), (x_{maxk} + R_{maxk})]$

Therefore, the interval $(x_k, x_k + w_k)$ takes the form as

$$([x_{mink}, x_{maxk}], [(x_{mink} + R_{mink}), (x_{maxk} + R_{maxk})])$$

We now define the join operation on two intervals $[x_{mink} + x_{maxk}]$ and $[(x_{mink} + R_{mink}), (x_{maxk} + R_{maxk})]$ to determine the final interval as

$$(x_{mink}, x_{maxk} + R_{maxk})$$

Similarly, interval $(x_i, x_i + w_i)$ takes the form

$$(x_{mini}, x_{maxi} + R_{maxi})$$

The y co-ordinate of a module i can hence be determined as follows: If $\psi(i)$ is non-empty, we have from equations 2.4 and 2.5

$$y_i = max_{k \varepsilon \psi(i)} y_k + h_k$$

Otherwise $y_i = 0$

where $y_k + h_k$ lies in the interval $[(y_{mink} + S_{mink}), (y_{maxk} + S_{maxk})]$

Hence, in the affine form,

$$y_i = max_{k \, \varepsilon \, \psi(i)} [(y_{mink} + S_{mink}), (y_{maxk} + S_{maxk})] \qquad (4.10)$$

Otherwise $y_i = 0$ (4.11)

The interval with the maximum value will be the one that has the maximum value of $((y_{maxk} + S_{maxk}) - (y_{mink} + S_{mink}))$

The algorithm for finding the placement from a corresponding B*-Tree is outlined in Figure 4.8. It uses a contour structure to reduce the run time for finding the *y* co-ordinate of a module while solving the equations (4.10) and (4.11).

```
Input: B*-Tree(π [0:n])
Output: Placement with position ($x_{mini}$, $x_{maxi}$) for each module $m_i$
Begin
    Set perm = 1
    Set contour = NULL
    Set current_contour = 0
    For code = 0 to n-1
        If code = 0 then
            Set current_module = π [perm]
            If current_contour = 0 then
                Set $x_{min}$[current_module] = $x_{min}$[current_contour] + $R_{min}$[current_contour]
                Set $x_{max}$[current_module] = $x_{max}$[current_contour] + $R_{max}$[current_contour]
            Else set $x_{min}$[curent_module] = $x_{max}$[current_module] = 0
            End if
            Set y[current_module] = find_max_y (contour, current_module)
            update_contour (contour, current_module)
            Set current_contour = current_module
            Set perm = perm + 1
        Else set current_contour = prev[current_contour]
        End if
    End For
End.
```

Figure 4.8: Pseudo code for determination of *x* and *y* co-ordinates using our approach.

find_max_y determines the *y* co-ordinate of current module according to equations (4.10) and (4.11) as explained above. All the operations are affine operations which have been discussed in detail in this section above.

4.3.2.3 Determination of Area Range

After determining the placement, i.e. the range of (*x*, *y*) co-ordinates for each block, we calculate the range of total area [A_{mini}, A_{maxi}] for a relative floorplan corresponding to each feasible B*-Tree.

To determine minimum area i.e. $A_{min,}$ we select the maximum value of (x_{mini} + R_{mini}) and (y_{mini} + S_{mini}) among all the blocks b_i . These values denote the *x* co-ordinate of the rightmost and *y* co-ordinate of the topmost blocks respectively in the floorplan layout. Thus, A_{min} can be calculated as below

$$A_{min} = max_{\ i\,=\,1\ to\ n}\,(x_{mini} + R_{mini}) \times max_{\ i\,=\,1\ to\ n}\,(y_{mini} + S_{mini}) \qquad (4.12)$$

Similarly for determining maximum area i.e. A_{max}, we select the maximum value of (x_{maxi} + R_{maxi}) and (y_{maxi} + S_{maxi}) among all the blocks b_i. A_{max} can be calculated as below

$$A_{max} = max_{\ i\,=\,1\ to\ n}\,(x_{maxi} + R_{maxi}) \times max_{\ i\,=\,1\ to\ n}\,(y_{maxi} + S_{maxi}) \qquad (4.13)$$

4.3.2.4 Determination of Wirelength Range

We determine the wirelength estimation by the half perimeter wirelength (HPWL) method as explained in chapter 2.1.2.1. Before the placement, pin locations are specified relative to the block's lower left co-ordinates. Absolute locations of pins are determined with respect to the origin of the chip after the placement. We use the information about the ranges of block locations determined from the placement to compute the ranges of pin locations and hence the wirelength range.

Each block b_i has lower left co-ordinates (x_i, y_i) in the ranges $[x_{mini}, x_{maxi}]$ and $[y_{mini}, y_{maxi}]$ respectively. Let the relative pin locations with respect to block's lower left co-ordinates, denoted as (x_{pi}, y_{pi}), lie in the ranges $[x_{minpi}, x_{maxpi}]$ and $[y_{minpi}, y_{maxpi}]$ respectively. Absolute pin locations, denoted as (x_{api}, y_{api}), are computed according to equations 2.6 and 2.7 as below.

$$x_{api} = x_i + x_{pi}$$

$$y_{api} = y_i + y_p$$

Expressing all the operands in affine form, we get

$$x_{api} = (x_{0i} + x_{0pi}) + (x_{ki}\varepsilon_{ki} + x_{kpi}\varepsilon_{kpi}) \qquad (4.14)$$

Similarly , $$y_{api} = (y_{0i} + y_{0pi}) + (y_{ki}\varepsilon_{ki} + y_{kpi}\varepsilon_{kpi}) \qquad (4.15)$$

where x_{api} and y_{api} lie in the ranges $[(x_{mini} + x_{minpi}), (x_{maxi} + x_{maxpi})]$ and $[(y_{mini} + y_{minpi}), (y_{maxi} + y_{maxpi})]$ respectively.

HPWL of a net *j* consisting of *k* number of pins is calculated according to equations 2.8 and 2.9 as below

$$HPWL_{xj} = max_{\,i=1\ to\ k}\,(x_{api}) - min_{\,i=1\ to\ k}\,(x_{api})$$

$$HPWL_{yj} = max_{\,i=1\ to\ k}\,(y_{api}) - min_{\,i=1\ to\ k}\,(y_{api})$$

Thus , $HPWL_j = HPWL_{xj} + HPWL_{yj}$

$$HPWL_j = max_{\,i=1\ to\ k}\,(x_{api} + y_{api}) - min_{\,i=1\ to\ k}\,(x_{api} + y_{api}) \qquad (4.16)$$

$(x_{api} + y_{api})$ lies in the range $[(x_{mini} + x_{minpi} + y_{mini} + y_{minpi}), (x_{maxi} + x_{maxpi} + y_{maxi} + y_{maxpi})]$. $max_{\,i=1\ to\ k}\,(x_{api} + y_{api})$ is given by the maximum value of $((x_{maxi} + x_{maxpi} + y_{maxi} + y_{maxpi}) - (x_{mini} + x_{minpi} + y_{mini} + y_{minpi}))$ among all such values for *k* pins.

Similarly, $min_{\ i\ =\ 1\ to\ k}\ (x_{api} + y_{api})$ is given by the minimum value of $((x_{maxi} + x_{maxpi} + y_{maxi} + y_{maxpi}) - (x_{mini} + x_{minpi} + y_{mini} + y_{minpi}))$ among all such values for k pins. Let u and v be the two pins for which maximum and minimum values are found respectively in the equation 4.16. Thus,

$$HPWL_{minj} = min\ ((x_{apu} + y_{apu}) - (x_{apv} + y_{apv})) \qquad (4.17)$$

$$HPWL_{maxj} = max\ ((x_{apu} + y_{apu}) - (x_{apv} + y_{apv})) \qquad (4.18)$$

Total wirelength is determined by summing up the *HPWL* of all nets.

$$WL = \sum_{j=1}^{N} HPWL_j$$

The maximum and minimum values of total wirelength are determined as follows,

$$WL_{min} = \sum_{j=1}^{N} HPWL_{minj} \qquad (4.19)$$

$$\text{and } WL_{max} = \sum_{j=1}^{N} HPWL_{maxj} \qquad (4.20)$$

Our algorithm for placement is summarized in Figure 4.9. The complexity of our algorithm is $O(n)$.

```
Begin
    For each $b_i$
        Convert $w_i$, $h_i$ and $(x_i, y_i)$ to affine form.
    End For
    Calculate $[x_{mini}, x_{maxi}]$ and $[y_{mini}, y_{maxi}]$ for each $b_i$ according to pseudo code in
    Figure 4.8
    Calculate $[A_{min}, A_{max}]$ according to equations 4.12 & 4.13
    Calculate $[WL_{min}, WL_{max}]$ according to equations 4.19 & 4.20
    Calculate $A_{mean}$ & $A_{range}$ and $WL_{mean}$ & $WL_{range}$ according to equations 4.4 to 4.7
End.
```

Figure 4.9: Pseudo code of placement using our approach.

4.4 Experimental Results

To evaluate our affine arithmetic based floorplanning algorithm, we performed a set of two experiments. First experiment was conducted using the Monte Carlo simulation approach. It presents the ranges of area and wirelength and the B*-Tree corresponding to the best feasible solution. Second experiment demonstrates the effectiveness of our algorithm in deriving the ranges of area and wirelength for the best feasible solution. We then compare the results of MC approach with those of our algorithm to verify its accuracy.

4.4.1 Experimental Setup

The experimental setup is as follows. The simulated annealing floorplanning algorithm is implemented in C++ programming language on an Intel Pentium 4, 1.73 GHz PC with 1 GB RAM. The operating system is RedHat Linux v6.1, kernel version 2.4.

The experiments were performed on a set of five MCNC benchmark circuits that consists of hard modules. We tested all these benchmarks. Table 4.6, on the next page, gives the information of MCNC benchmarks. Since the benchmarks contain modules with fixed dimensions, we adopt a procedure to convert fixed dimensions into variable dimensions for use in our floorplanning algorithm. We calculate the ranges of width and height of each block to be within $\pm$ 2% of the given fixed dimensions for width and height in each MCNC benchmark circuit. However, any value typically in the range 0-5% can be provided depending on the impact of design uncertainty on each block.

Table 4.6. MCNC benchmarks information.

Circuit	Block #	Net #	Pin #	Pad #
apte	9	97	214	73
xerox	10	203	696	2
hp	11	83	264	45
ami33	33	123	480	42
ami49	49	408	931	22

In the simulated annealing process, the temperature was decreased at a constant rate (0.9). We terminate the annealing process if the rejection rate of moves exceeds the convergent rate of 0.85 at a certain temperature or the temperature decreases beyond the threshold value of 0.1.

We consider area optimization as the main criterion in the cost function to obtain floorplans with the smallest range and average values of area possible for all the benchmarks. To balance the accuracy and run time, we chose to run 10,000 iterations for the Monte Carlo simulation.

4.4.2 Results and Analyses

Table 4.7 shows the results for area and wirelength for MC approach. For each test case, the mean and range values are listed together with the minimum and maximum values. The run time for MC approach is also provided.

Table 4.7. Results for area and wirelength using MC approach.

Circuit	A_{mean} (mm^2)	[A_{min}, A_{max}] (mm^2)	A_{range} (mm^2)	WL_{mean} (mm)	[WL_{min}, WL_{max}] (mm)	WL_{range} (mm)	Run Time (min)
apte	47.96	[46.61, 49.31]	2.70	450.84	[443.78, 457.9]	14.12	15.78
xerox	9.98	[9.25, 10.71]	1.46	270.18	[268.13, 272.22]	4.09	23.05
hp	20.63	[19.94, 21.31]	1.37	471.16	[462.23, 480.08]	17.85	49.85
ami33	1.21	[1.20, 1.22]	0.02	86.23	[85.03, 87.43]	2.40	97.41
ami49	38.64	[37.54, 39.74]	2.20	1075.14	[1059.76, 1090.51]	30.75	167.24

Table 4.8 shows the results for area and wirelength for our algorithm approach. The run time for our algorithm is also provided.

Table 4.8. Results for area and wirelength using our approach.

Circuit	**A_{mean} (mm^2)**	**[A_{min}, A_{max}] (mm^2)**	**A_{range} (mm^2)**	**WL_{mean} (mm)**	**[WL_{min}, WL_{max}] (mm)**	**WL_{range} (mm)**	**Run Time (sec)**
apte	48.15	[46.6, 49.7]	3.10	458.13	[450.01, 466.24]	16.23	5.92
xerox	10.09	[9.25, 10.92]	1.67	274.61	[272.42, 276.79]	4.37	7.47
hp	20.84	[20.07, 21.60]	1.53	484.34	[474.29, 494.39]	20.10	32.32
ami33	1.22	[1.20, 1.23]	0.03	89.98	[88.33, 91.63]	3.30	57.72
ami49	38.88	[37.68, 40.08]	2.40	1098.41	[1081.64, 1115.17]	33.53	67.08

We can see that the run time of our algorithm on all test cases is very fast. The circuit with the longest run time, *ami49*, was analyzed in only about 67 seconds while the MC simulation required 167 minutes.

Table 4.9 shows a comparison of the results for area and wirelength of MC approach with those of our algorithm. The results of our algorithm can be seen to be very close to the MC results: the average error for area is –0.78% for the mean and –12.96% for the range. The average error for wirelength is –2.43% for the mean and –13.23% for the range.

Table 4.9. Area and wirelength comparison results of our algorithm and Monte-Carlo simulation (MC) method.

E = (MC − Our) / MC %				
Circuit	**A_{mean} E (%)**	**A_{range} E (%)**	**WL_{mean} E (%)**	**WL_{range} E (%)**
apte	-0.39	-0.13	-1.59	-13.00
xerox	-1.09	-12.57	-1.61	-6.40
hp	-1.00	-10.45	-2.72	-11.19
ami33	-0.81	-33.33	-4.14	-27.27
ami49	-0.61	-8.33	-2.11	-8.29

We observe that the results of mean and range from our approach are larger than those from MC approach for both area and wirelength. This negative value of error can be explained by considering the fact that MC approach typically evaluates the model based on computations on the actual values obtained by random generation whereas our approach predicts the range of values by mathematical modeling. Also, AA accounts for the worst case of the simulation. However, worst case scenario is seldom reached in real situations. Hence, the interval obtained in AA is slightly over-pessimistic.

We also observe that the error for mean is smaller than the error for range both for the area and wirelength. This is due to the fact that Monte Carlo simulations do not cover the worst case scenarios correctly, and hence the larger error in range estimation. However, mean values are well matched to those from our algorithm since they are based on typical values normally encountered in real simulations.

Above results demonstrate the accuracy and effectiveness of our algorithm based on the affine arithmetic model to estimate the area and wirelength ranges of a floorplan in the presence of dimension variations.

4.4.3 Discussion

Although we have chosen to perform our experiments with the consideration of area minimization as the main criterion in the cost function, we obtain similar behavior in the results pattern if we consider both area and wirelength minimizations as the objective. Since the goal of our floorplanning algorithm is to predict the ranges of area and wirelength under the variations in dimensions of modules, we can obtain the results for any combination of weights assigned to α and β in the cost function.

4.5 Conclusion

In this chapter, we have presented our variability-aware floorplanner. The problem has been formulated as a floorplanning optimization problem under variations in dimensions for each module resulting due to design uncertainty in early chip planning stage. An example has been provided to make the problem clearer. Then, we have explained the entire methodology followed in developing the MC based algorithm and our algorithm. In our algorithm, we have explained the procedure for getting the placement from a corresponding B*Tree using the contour structure. Finally, the experimental results presented prove the effectiveness of our algorithm in determining the ranges of area and wirelength under variations in dimensions of each module. We compare the results of our algorithm from MC approach and show that it performs better than MC approach in terms of run time by testing on a set of MCNC benchmark circuits.

Chapter 5

Conclusions and Future Directions

5.1 Conclusions

With the aggressive scaling of process technologies towards the deep submicron region, increased levels of integration within a singe die have imposed rigid constraints on the power budget and hence the temperature estimations of a chip. Also, the challenging increase in the amount of variability due to uncertainty in block and interconnect physical parameters across early phases of design has led to the need for variability-aware mechanisms that can correctly model these variations. In this thesis, we have discussed two significant works related to the area of interconnect power and thermal aware floorplanning, and variability-aware floorplanning.

In the first part of the thesis, we have shown how to improve the temperature distributions of a chip and reduce hotspots when considering heavily used interconnect circuits through interconnect power and thermal aware floorplanning. We have presented an interconnect power and thermal aware floorplanner that takes into account the effects of the switching activity of interconnects in deriving power consumption in estimating the peak temperatures. We have demonstrated that the peak temperatures can be underestimated by as much as 15^{o}C without including switching activity of interconnects in determination of power. Finally, we have shown the effectiveness of our floorplanner in reducing peak temperatures by as much as 20% using MCNC benchmarks with a comparable area and a penalty of 2% in terms of the total wirelength. Chip temperatures are expected to further increase in future designs based on deep sub-micron technology and heavy interconnect usages, thus making the benefits of interconnect power and thermal aware floorplanning even more prominent.

In the second part of the thesis, we have developed a variability-aware floorplanning algorithm. Variability issues at floorplanning stage have not been studied with much importance upto now. We have presented a B*-Tree based floorplanning algorithm based on affine arithmetic to determine the ranges of chip area and wirelength in the presence of variations in dimensions of each module. We have verified the accuracy of the method with Monte Carlo simulation. The average errors of mean and standard deviation values computed by the proposed method are −0.78% & −12.96% respectively for area, and −2.43% & −13.23% for wirelength respectively by testing on five MCNC benchmarks. The fast run time of our algorithm proves its superiority over Monte Carlo simulation approach. Further increase in uncertainty and hence variability effects in future complex designs will make the benefits and significance of our affine arithmetic based floorplanning algorithm even more prominent.

5.2 Future Directions

Since the increase in variations in block and interconnect parameters pose a challenge to the performance analysis of high-speed designs, many other important parameters like power dissipation and temperature of the chip will also be drastically impacted. As we have discussed in our work, power dissipation and temperature has become more crucial with shrinking technologies and need to be modeled accurately. With the combined effect of variability impacts, these parameters will become of paramount importance and will need to be studied in greater detail in early phases of chip planning i.e. design prototyping.

Thus, the work done in our thesis opens door for future research in the area of variability-aware floorplanning taking into account thermal and power dissipation of device and interconnects. Since design uncertainty affects the power and temperature of the chip besides the timing of circuits considerably, we need to predict the ranges of these parameters as well, under the presence of variations in dimensions of modules. The projected future work will be based on combining the power dissipation effects with the variations to yield variable ranges of temperature and power dissipation besides the chip area and wirelength.

The projected future work will estimate the range of power dissipation for each functional block by taking into account the range of its area. Similarly, range of power dissipation of each interconnect will be determined by taking into account the range of its netlength. Range of area of each block and range of netlength for each interconnect is already determined in our variability-aware floorplanner. Further, it will compute the range of power distribution profile of the chip by considering the variations in netlength and area as mentioned above. Range of temperature estimations of the chip will be determined using the range of power dissipation values for each block and the ranges of floorplan metrics like area and wirelength.

Thus, our work done in this thesis provides a promising framework for significant future work in the direction of computing impacts of design uncertainty on power dissipation and temperature estimations of the chip at the floorplanning stage.

Bibliography

[1] J. Srinivasan, S.V. Adve, P. Bose, and J. Rivers. “The Impact of Technology Scaling on Lifetime Reliability”. In Proc. of International Conference on Dependable Systems and Networks, June 2004.

[2] K. Bazargan, S. Kim and M. Sarrafzadeh. “NOSTRADAMUS: A Floorplanner of Uncertain Designs”. In Proc. ISPD, 1998.

[3] Goldberg, D.E. Genetic Algorithms in Search, Optimization, and Machine Learning, New York: Addison-Wesley, 1989.

[4] R.H.J.M. Otten. “Automatic Floorplan Design”. In Proc. DAC, 1992, pp.261-267.

[5] D. F. Wong, and C. L. Liu, “A New Algorithm for Floorplan Design”. In Proc. DAC, 1986, pp.101-107.

[6] P.N. Guo, C.K. Cheng, and T. Yoshimura. “An O-Tree Representation of Non-Slicing Floorplan and Its Applications”. In Proc. 36th DAC, June 1999, LA, U.S.A, pp. 268-273.

[7] X. Tang, and D.F Wang. “FAST SP: A Fast Algorithm for Block Placement based on Sequence Pair”. In Proc. ASP-DAC, 2001, pp 521-526.

[8] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. “Module Placement on BSG-Structure and IC Layout Applications”. In Proc. ICCAD, Nov. 1996, CA, U.S.A., pp. 484-491.

[9] T.C. Wang, and D.F. Wong. “An Optimal Algorithm for Floorplan and Area Optimization”. In Proc. DAC, 1990, pp.180-186.

[10] Y.C. Chang, Y.W. Chang, G.M. Wu, and S.W. Wu. “B*-Trees: A New Representation for Non-Slicing Floorplans”. In Proc. 37th DAC, June 2000, CA, U.S.A., pp. 458-463.

[11] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. “VLSI module placement based on rectangle-packing by the sequence pair”. In IEEE Trans. Computer-Aided Design, Dec. 1996, vol. 15, pp. 1518-1524.

[12] X. Hong, G. Huang, T. Cai, J. Gu, S. Dong, C.K. Cheng, and J. Gu. “Corner Block List: An effective and efficient topological representation of non-slicing floorplan”. In Proc. ICCAD, Nov. 2000, CA, U.S.A, pp. 8-12.

[13] J.M. Lin and Y.W. Chang. “TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans”. In Proc. 38th DAC, June 2001, NV, U.S.A., pp. 764-769.

[14] Thermal-aware 3D Microarchitectural Floorplanning. A Technical Report, May 20,

2005, http://www.cs.virginia.edu/ techrep/CS-2005-08.pdf.

[15] Y. Han, I. Koren and C.A. Moritz. “Temperature Aware Floorplanning”. In Second Workshop on Temperature-Aware Computer Systems, 2005.

[16] W.L. Hung, C. Addo-Quaye, T. Theocharides, Y. Xie, N. Vijaykrishnan, and M. J. Irwin. “Thermal-aware floorplanning using genetic algorithms”. In Proc. ISQED, 2005.

[17] J. Cong, J.Wei, and Y. Zhang. “A Thermal-Driven Floorplanning Algorithm for 3D ICs”. In Proc. ICCAD, 2004.

[18] C. Chu and D.F. Wong. “A matrix synthesis approach to thermal placement”. In Proc. ISPD, 1997.

[19] Brent Goplen and Sachin Sapatnekar. “Efficient Thermal Placement of Standard Cells in 3D ICs using a Force Directed Approach”. In Proc. ICCAD, 2003.

[20] W.L. Hung, G.M. Link, Yuan Xie, N. Vijaykrishnan, and M. J. Irwin. “Interconnect and Thermal-aware Floorplanning for 3D Microprocessors”. In Proc. 7th ISQED, 2006.

[21] J. Cong and D.Z. Pan. “Interconnect performance estimation models for design planning”. In IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, June 2001, pp. 739-752.

[22] P. Kapur, G. Chandra, and K.C. Saraswat. “Power estimation in global interconnects and its reduction using a novel repeater optimization methodology”. In Proc. DAC, 2002.

[23] D. Sylvester and K. Keutzer. “Getting to the bottom of deep submicron”. In Proc. ICCAD, 1998.

[24] W. Liao and L. He. “Full-chip interconnect power estimation and simulation considering concurrent repeater and flip-flop insertion”. In Proc. ICCAD, 2003.

[25] P. Prabhakaran and P.Banerjee. “Simultaneous Scheduling, Binding and floorplanning for Interconnect Power Optimization”. In Proc. 12th International Conference on VLSI Design – ‘VLSI for the Information Appliance’, 1999, pp. 423.

[26] R.mehra, L.M Guerra and J.M Rabeay. “Low power Architectural synthesis and the impact of exploiting locality”. Journal. VLSI signal processing, vol. 13, No. 8, pp. 877-88, Aug 1996.

[27] N. S. Kim, T.M. Austin, D. Blaauw, T.N. Mudge, K. Flautner. J. S. Hu, M.J. Irwin, M.T. Kandemir, and N. Vijaykrishnan, “Leakage current: Moore's law meets static power”. In IEEE Computer, vol. 36, no. 12, pp. 68-75, 2003.

[28] S.M. Martin, K. Flautner, T. Mudge, and D. Blaauw. “Combined dynamic voltage

scaling and adaptive body biasing for lower power microprocessors under dynamic workloads". In Proc. ICCAD, 2002, pp. 721-725.

[29] K. Skadron, T. Abdelzaher, and M. Stan. "Control-Theoretic Techniques and thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management". In Proc. HPCA 2002.

[30] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao,K. Gala, and R. Panda. "Statistical delay computation considering spatial correlations". In Proc. ASP-DAC, Kitakyushu, Japan, Jan. 2003, pp. 271-276.

[31] M. Berkelaar. "Statistical delay calculation, a linear time method" (Personal communication).

[32] M. Orshansky and K. Keutzer, "A general probabilistic framework for worst case timing analysis". In Proc. ACM/IEEE DAC, June 2002, pp. 556-561.

[33] S. Tsukiyama, M. Tanaka, and M. Fukui. "A statistical static timing analysis considering correlations between delays". In Proc. ASP-DAC, Jan. 2001, pp. 353-358.

[34] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula. "Computation and refinement of statistical bounds on circuit delay". In Proc. DAC, June 2003, pp. 348-353.

[35] J. Liou, K. Cheng, S. Kundu, and A. Krstic. "Fast statistical timing analysis by probabilistic event propagation". In Proc. ACM/DAC, June 2001, pp. 661-666.
[36] R.E Moore, Interval Analysis, prentice Hall, 1966.

[37] J. Stolfi and L.H. de Figueiredo. "An introduction to affine arithmetic". TEMA Tend. Mat. Apl. Computing, 4, No. 3 (2003), 297-312.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". Science, 1983, 220, (4598), pp. 671-680.

[39] G.S. Fishman. Monte Carlo: Concepts, Algorithms, and Applications, Springer-Verlag, 1995.

[40] V. Axelrad and J. Kibarian. "Statistical aspects of modern IC designs". In Proc.28th European Solid-State Device Research Conference, Bordeaux, France, Sept. 1998, pp. 309-321.

[41] http://www.isonics.com

[42] Naveed A. Sherwani. Algorithms For VLSI Physical Design Automation. Kluwer Academic Publishers, 3rd edition, 1999.

[43] M. Sarrafzadeh, C.K. Wong. An Introduction to VLSI Physical Design. McGraw-Hill, 1996.