Bus-Driven Floorplanning

LAW Hoi Ying

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science & Engineering

Abstract

As technology advances, the complexity of VLSI circuit design grows rapidly. Interconnect-driven floorplanning has become a major concern in modern floorplanning. Bus is a collection of wires running over a set of modules. It is favorable to align the set of modules that a bus goes through in such a way that routing can be done easily. In this thesis, the bus-driven floorplanning problem in 2D and 3D chips is considered. Besides, a 3D floorplan representation is proposed to solve the 3D floorplanning problem.

The bus-driven floorplanning problem involves the placement of blocks and buses. Given a set of blocks and bus specifications (the width of each bus and the blocks that the bus need to go through), we will generate a floorplan solution such that all the buses go through their blocks in less than or equal to 2-bends, with the area of the floorplan and the total area of the buses minimized. The approach proposed is based on a simulated annealing framework. Using the sequence pair representation, we derived and proved some necessary conditions for feasible buses, for which we allow 0-bend, 1-bend, or 2-bend. Then, we check whether there are buses that cannot be placed at the same time. Finally, a solution is generated giving the coordinates of the modules and the buses. Comparing with the most updated previous work by Xiang et al., our algorithm can handle buses going through many blocks and the dead space of the floorplan obtained is also reduced.

3D chips are useful in reducing interconnect lengths. However, there is not much previous work done in 3D floorplanning. In this thesis, we have proposed a 3D floorplan representation called *Layered Transitive Closure Graph (LTCG)*, based on the Transitive Closure Graph (TCG) representation for non-slicing floorplans, in addition with some layer information. A method is introduced to align blocks (of the same bus) on different layers. A floorplanner is implemented using the LTCG representation. Experimental results have shown that LTCG is a promising representation for 3D floorplans and can handle bus planning in 3D floorplan effectively.

摘要

隨著科技進步,超大規模集成電路的複雜性正在迅增長。互連電路主導的佈局規劃成為了現代佈局的一個重要課題。匯流排是一組需要經過數個組件的電線。如果那些組件在佈局規劃時已被排列成可以讓匯流排容易通過的序列，會令到整個設計過程更流暢。在本論文中，我們將會發表一個有效的匯流排主導佈局規劃方案。此外，由於三維電路可以有效解決互連電路的問題，我們將會發表一個三維的佈局規劃表示法。

匯流排主導佈局規劃問題牽涉組件和匯流排。已知一組組件和匯流排規格(包括匯流排的寬度和匯流排須要經過哪些組件)，我們要計算出一個可以讓所有匯流排經過組件的佈局規劃, 而匯流排最多可以屈曲兩次，同時要令整個佈局規劃的面積縮到最少。我們提出的方案用了模擬降溫法。根據序列組的表示法，我們衍生出和證明了一些可行匯流排的必要條件，而我們只允許匯流排零屈曲、一次屈曲、或是兩次屈曲。然後，我們會檢查是否有些匯流排不能同時存在。最後，我們會得到一個包含所有組件座標和匯流排座標的佈局規劃。與Xiang等提出的方案發表的結果比較，我們的方案可以處理一些經過很多組件的匯流排，而無效位置也比他們的少。

三維晶片對於減少互連電線長度很有效。然而，在這個範疇沒有很多前人的工作。在這論文中，我們根據Transitive Closure Graph (TCG)提出了一個三維的佈局規劃表示法名為Layered Transitive Closure Graph (LTCG)，加入了層的資訊。我們提出了一個方法去排列同一個匯流排在不同層要經過的組件。我們研究了一個佈局規劃配置器。實驗結果證明了LTCG是一個大有可為的三維佈局規範表示法，而且能夠效率地處理匯流排。

Acknowledgments

First of all, I must thank my supervisor, Professor Evangeline Fung Yu Young. She has helped me a lot in my research. Throughout the two years studies, she has given me guidances, encouragements, ideas, and advices. She is considerate and understanding. Without her, I am not possible to finish this work.

I would like to express my thanks to my marker, Professor David Yu Liang Wu. He has given me invaluable advices and constructive suggestions, which are very helpful for improvement of my work.

I would also like to thank the authors of [1] who have kindly given us their program and test cases, such that we can conduct the experiments and compare our results with theirs.

Finally, I would like to thank my colleagues. They have supported me and bolstered me. Without them, I would not have such an enjoyable and memorable school life in my memory.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The technology of *integrated circuit (IC)* was widely adopted for computing devices like microprocessors, memory modules, and many other interface chips since 1960s. It is not surprising to find that we are surrounded by a huge number of computing devices in daily life, such as our personal computers, the ATM machines we use to withdraw cash, and many other electronic appliances. IC is one of the core components of those computing facilities.

As the *Very Deep Sub-Micron (VDSM)* technology advances, IC has evolved from *Small Scale Integration (SSI)* to *Very Large Scale Integration (VLSI)*. The former consists of a few transistors only, where the latter consists of billions of transistors. According to Moore's Law [5], it was predicted that the number of transistors in a single IC will double in every 1.5 years. Table 1.1 show the predicted technology roadmap from 1997 to 2009 [3]. In the foreseeable future, the technology of VLSI will continue to scale down, to produce faster, more complicated yet more powerful ICs. As a side effect, the interconnections will hence become longer and denser, and it will be desirable to keep the sizes of the chips as small as possible. This growing trend has brought many new challenges to VLSI design automation, and make the design process more difficult and complicated.

Technology (μm)	**0.25**	**0.18**	**0.15**	**0.13**	**0.1**	**0.07**
Year	1997	1999	2001	2003	2006	2009
Number of Transistors	11M	21M	40M	76M	200M	520M
Across Chip Clock (MHz)	750	1200	1400	1600	2000	2500
Area (mm^2)	300	340	385	430	5.20	6.20
Wiring Levels	6	6-7	7	7	7-8	8-9

Table 1.1: Technology Roadmap [3].

Producing a tiny chip is a time consuming process. There are many steps to go through, and many of them are computationally expensive. Many algorithms have been developed in CAD (Computer Aided Design) tools to help accomplishing the task, but there are still many unresolved problems and new challenges to be explored. In the following sections, the VLSI design cycle and the physical design cycle will be described briefly. After that, the floorplanning problem will be introduced and discussed.

1.1 VLSI Design Cycle

To design a VLSI circuit, a series of steps has to be gone through. The process starts with a formal specification, and the final product is a fabricated chip. Figure 1.1 shows a VLSI design cycle. In this section, the key steps leading to a packaged chip will be described briefly.

System Specification

The first step in the design cycle is to prepare a formal specification of the system. This specification should state clearly the performance, functionality, physical dimension, power consumption, and other requirements of the VLSI system. Once the specification is laid down, the design process can proceed and the requirements stated has to be satisfied.

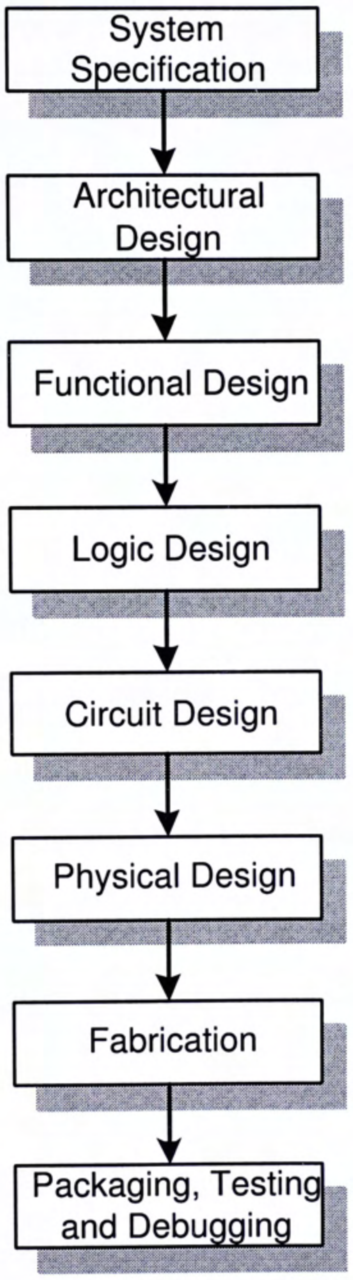


Figure 1.1: The VLSI Design Cycle.[2]

Architectural Design

Architectural decisions will be made in this step. For example, whether RISC (Reduced Instruction Set Computer) or CISC (Complex Instruction Set Computer) will be adopted, the number of ALUs (Arithmetic Logic Unit) or floating point units, or the number and structure of pipelines. After architectural design, engineers can predict the performance or power consumption of the system accurately. The prediction can help determining whether the design is likely to meet the specification.

Functional Design

In functional design, the behavior of the system, in terms of input, output, and timing requirement, will be specified, in which the internal structure is not concerned. The behavior of a system refer to the functionality that the system is capable of. Besides, interconnections between different units will also be defined in this step.

Logic Design

In this step, logic operations that represent the functional design of the system are derived and tested. Boolean expressions will be used to describe the logic operations. The logic operations include the control flow, arithmetic operations, and register allocation. The logic design has to be conformed to the functional design, and will be simulated to verify its correctness.

Circuit Design

Based on the logic design, a circuit representation can be derived. The circuit representation is a detailed circuit diagram. It shows clearly the cells, gates, transistors, and other circuit elements, together with the interconnections between them. During the design process, the speed and power requirements are also taken into account.

Physical Design

The step of transforming a circuit representation into a geometric representation is called physical design. The geometric representation of a circuit is called a *layout*. During physical design, problems like where to place the modules, how the interconnections between the modules should be made etc., will be addressed. As physical design is a crucial yet complex step in the design cycle, it can be further broken down into sub-steps, such as partitioning, floorplanning, placement, routing, and compaction.

Fabrication

Once the layout is produced and verified, it is ready for fabrication. The layout data is converted into photo-lithographic masks. There are several steps of the fabrication process, including deposition and diffusion of various materials on the wafer. A large wafer can be used to produce many chips. A prototype is made before the mass production of a chip.

Packaging, Testing and Debugging

The fabricated chip is tested in this step. Each chip is tested to ensure that all the requirements in the specification are met, and it can function properly. After that, the chips will be mass produced and packaged.

1.2 Physical Design Cycle

As mentioned before, a circuit representation will be transformed into a layout in the physical design step. It is usually broken down into several sub-steps. A physical design cycle is shown in Figure 1.2. The details of each step will be discussed in this section.

Partitioning

In order to achieve complicated functionalities, a chip may actually be comprised of millions of transistors. Breaking down a big problem into smaller sub-problems is always a good strategy to solve complicated problems. As huge circuits are hard to be managed efficiently and cannot be layout all at once, decomposition into finer sub-systems is a must in the design cycle. The step of decomposition is called partitioning, and the sub-circuits partitioned are called *blocks*. After partitioning circuits into blocks, each of them can then be designed effectively, independently, and simultaneously so as to ease the design process. Factors like the block sizes, block dimensions and interconnections between different blocks should be taken into account.

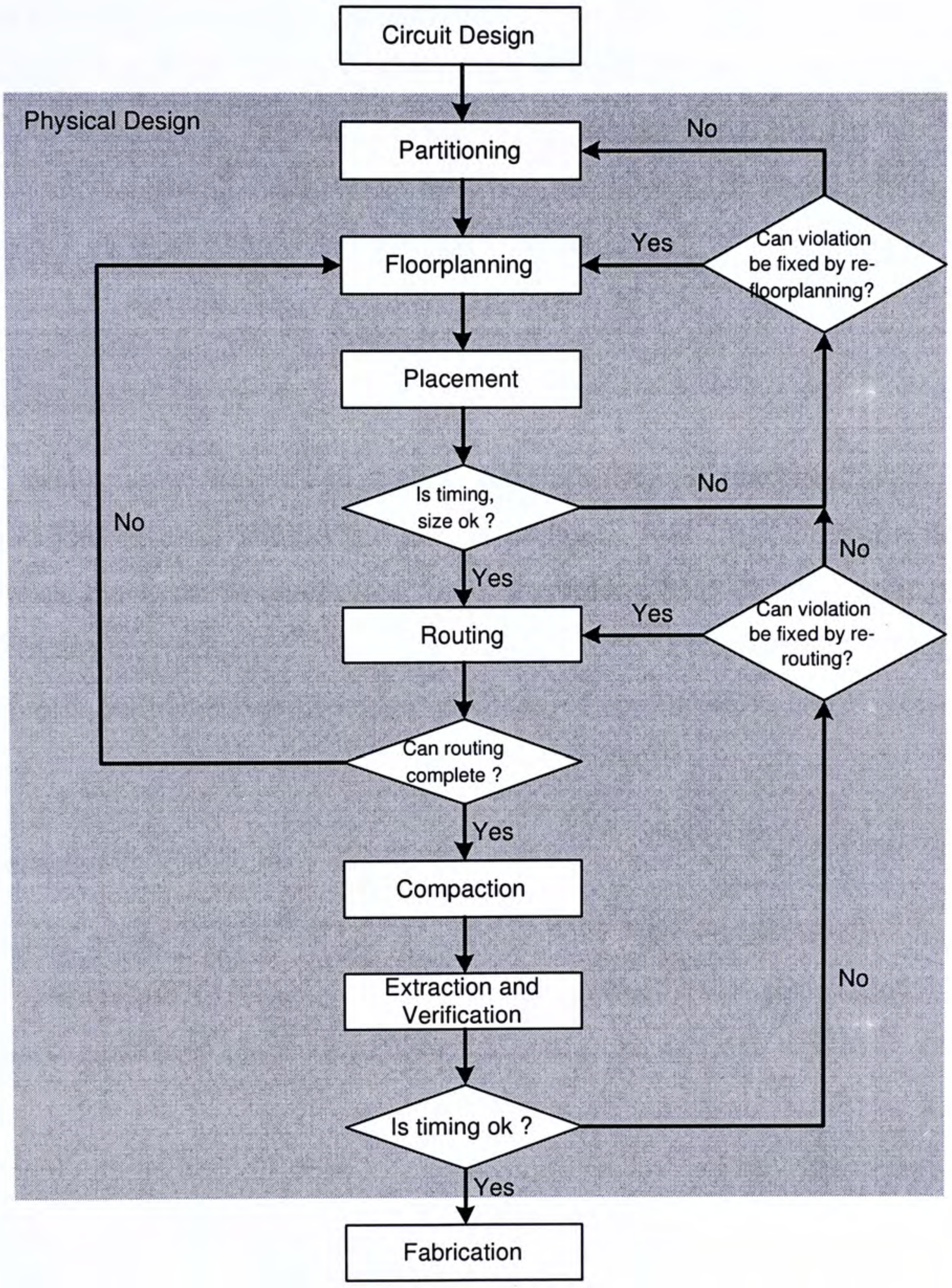


Figure 1.2: A Physical Design Cycle.[2]

Floorplanning

During the step of floorplanning, the blocks are positioned on the chip roughly, so as to optimize the circuit size and performance according to the circuit specification. A compact design is favorable, but there are many other important aspects that have to be taken care of. For example, issues like the block dimensions and overall delay should be taken into account. In floorplanning, the decisions on block shapes and pin positions are made.

Placement

The exact positions of the blocks are determined in the placement step. The layout should meet the performance constraints, allow the interconnections between blocks to be made, and meet the timing goal. Floorplanning and placement are vital to the design process as it affects the ultimate design significantly and determines whether the required specifications can be met.

Routing

Routing means completing the interconnections between blocks according to the specified netlist. The space not occupied by the blocks, the routing space, is partitioned into channels and switchboxes. Connections are made within them. Routing can be further broken down into two phases, namely global routing and detailed routing.

1. **Global Routing:** Planning different routes from a global point of view, without fixing the exact path of each route yet. It is a rough plan to check whether completion of all interconnections is possible.

2. **Detailed Routing:** Complete each connection by computing the exact positions of the wires on the metal layers. After detailed routing, the geometric layouts of all the nets will be known.

There may be cases that some of the connections are not able to be routed. In those situations, the technique *rip-up and re-route* will be used, which means removing some of the routed connections and re-routing them in a different order. If the problem cannot be solved by this technique, engineers may need to go back to the earlier design phases in the physical design cycle, or even to the logic design step and start the whole process all over again.

Compaction

Compaction means making the chip design as small as possible. During this step, the layout is compressed from different directions so as to reduce the total area. Note that during compaction, it is necessary to ensure that no design rules or constraints are violated.

Extraction and Verification

The layout is verified in this step, to ensure that all the design rules and performance constraints are satisfied, before proceeding to the fabrication step. Design rules, such as wire separation rule, which is the minimum separation between two adjacent wires, have to be fulfilled. Besides, the functionality of the layout is also verified. If problem is found, engineers may need to go back to the earlier designing steps to fix the problem.

1.3 Floorplanning

As technology advances into the deep submicron era, circuit sizes and complexity increase dramatically. A good planning in the early design phase is crucial, in order to avoid unnecessary iteration in the design cycle. Floorplanning has become an important step in the physical design cycle.

The input to the floorplanning phase is a set of blocks, the area of each block, the possible shapes of each block, the number of terminals of each block, and the interconnections between blocks. In the floorplanning phase, we are going to plan the position and shape of each block, together with the pin positions. The shapes for some blocks are fixed and cannot be altered. We called those blocks *hard blocks*. For other blocks, the shapes can be altered as long as they are within the pre-set aspect ratios. Those blocks are called *soft blocks*. A formal definition of the floorplanning problem is given as followed:

Definition 1.1 The problem **floorplanning** is defined as:

Given a set of n modules $\{M_1, M_2, \cdots, M_n\}$, where each module M_i is associated with an area A_i, together with two aspect ratio bounds r_i and s_i such that $r_i \leq h_i/w_i \leq s_i$, where h_i and w_i is the height and the width of module i respectively. The output of the problem is a packing of the set of modules, i.e. the x- and y-coordinates and the dimension $(h_i,\ w_i)$ of each module. There should be no overlapping between modules, and the circuit performance should be optimized.

In this section, some floorplan objectives will be discussed. Besides, some approaches adopted today to solve the floorplanning problem will be presented.

1.3.1 Floorplanning Objectives

There are several objectives to be optimized in floorplanning, like the total chip area, the total wire length, the critical path delay etc. In this section, some common floorplanning objectives will be discussed.

Chip Area

Area minimization is one of the most commonly adopted objectives . Minimizing the chip area implies minimizing the wire length, and hence reducing the circuit delay.

Total Wire Length

In addition to minimizing the chip area, minimizing the total wire length directly is also another important goal. Beside the timing issues, using less wires to connect the modules means consuming less resources, and thus reducing the production cost.

Delay

In some cases, minimizing the total wire length is not enough. Timing is an important issue. The final circuit performance can be optimized by minimizing the delay on the critical path.

Routability

Routability refer to the possibility of completing all the connections. A non-routable floorplan is of no use even if it is area-optmized and delay-optimized.

Enhancing the routability of a floorplan means to reduce the chance of encountering routing problems in the downstream designing steps.

Others

There are still some other objectives in floorplanning, like minimizing heat dissipation, minimizing power consumption, etc. In our work, we focus on the bus-driven floorplanning problem to minimize interconnect delay by arraying the modules on the same bus in such a way that routing can be done effectively.

1.3.2 Common Approaches

The floorplanning problem is proved to be NP-complete. Thus, different heuristics are developed to solve the problem, which includes analytical approach, simulated annealing, genetic algorithm, force directed approach, constraint based approach, and other stochastic searching approaches.

Analytical Approach

In 1991, the author of [6] proposed that the floorplanning problem can be formulated as a *mixed integer linear program (MILP)*, such that the objective is a linear function, all constraints are linear functions, and some variables are real numbers while others are integers. However, the MILP problem itself is a NP-complete problem, and the run time of the best known algorithm is exponential to the number of variables and equations. Thus, this modelling can only solve problems of small scales. In 1998, a convex formulation [7] is proposed to reduce the number of variables and constraints used, by handling the aspect ratios of the blocks in an indirect way.

Simulated Annealing (SA)

Simulated Annealing is a widely adopted heuristic to solve NP-complete problem. It belongs to the probabilistic and iterative class of algorithms. The algorithm was originally proposed in [8] for finding the equilibrium configuration of a collection of atoms at a given temperature. The idea of using SA as an optimization tool is introduced in [9]. After that, it is suggested in [10] that SA can be used as a general technique for different optimization problems. This technique is used in [11][12][13][14][15] to solve the floorplanning problem.

SA mimics the process of metal cooling and freezing into a highly ordered crystalline structure with minimum energy (the annealing process). The framework of a simulated annealing based floorplanner can be described as follows: each floorplan in the solution space is represented by a representation (e.g., sequence representation, o-tree, etc.). The quality of each candidate floorplan is evaluated according to a cost function, which may take area, wirelength, etc. into consideration. The process starts with an initial solution x_0 and an initial temperature T_0. In each iteration, the candidate solution is changed a little, and is evaluated by the cost function. If the newly formed solution is better than the old one, it is accepted. Otherwise, the solution is accepted according to a probability depending of the temperature. If the temperature is high, the chance of accepting a worse solution will also be high. The temperature T will be cooled down at a cooling rate c. Finally, the process will terminate when the temperature is lower than a threshold T_t. The pseudo code is shown in Figure 1.3.

```
SIMULATED_ANNEALING ($ITER$, $T_0$, $T_t$, $c$)
1 $x \leftarrow x_0$
2 $T \leftarrow T_0$
3 WHILE $T \geq T_t$
4     FOR $i$ from 1 to $ITER$
5         $x_{new} \leftarrow$ move($x$)
6         $\Delta f \leftarrow$ cost($x_{new}$) - cost($x$)
7         $r \leftarrow$ random number between 0 to 1
8         IF $\Delta f < 0$ OR $r < \exp(-k\Delta f/T)$
9             $x \leftarrow x_{new}$
10        END IF
11    END FOR
12    $T \leftarrow T \times c$
13 END WHILE
14 RETURN $x$
```

Figure 1.3: Pseudo Code of Simulated Annealing.

Genetic Algorithm

Genetic algorithm [16][17] is another stochastic searching approach to solve NP-complete problems. A pseudo code of the general genetic algorithm approach is described in Figure 1.4. The process starts with a set of initial solutions namely population. By using two types of genetic operators, mutation and crossover, better populations can be obtained iteratively by means of evolution. Mutation means modifying one solution by applying a small change to itself. Crossover means forming a new solution by combing two solutions in the population.

1.3.3 Interconnect-Driven Floorplanning

Traditional floorplanners [18][11][13][17][19][20][21][22] aim at minimizing the chip area so as to increase the yield. However, as technology advances, the number of transistors and the number of interconnections involved increase dramatically. Interconnections between modules become longer and denser.

```
GENETIC_ALGORITHM ($P$, $R_c$, $R_m$)
1 $X \leftarrow \{x_1, x_2, \cdots, x_P\}$
2 WHILE stopping criteria not met
3     $X_{new} \leftarrow \phi$
4     WHILE number of children created $< P \times R_c$
5         select two solutions $x_i$ and $x_j$ from $X$
6         $x_{new} \leftarrow$ crossover($x_i$, $x_j$)
7         $X_{new} \leftarrow X_{new} \cup \{x_{new}\}$
8     END WHILE
9     select $P$ solutions from $X \cup X_{new}$, and call it $X$
10    WHILE number of children mutated $< P \times R_m$
11        select a solution $x_k$ from $X$
12        $x_{new} \leftarrow$ mutate($x_k$)
13        $X_{new} \leftarrow X_{new} \cup \{x_{new}\}$
14    END WHILE
15    $X \leftarrow X_{new}$
16 END WHILE
17 RETURN the best solution in $X$
```

Figure 1.4: Pseudo Code of Genetic Algorithm.

According to [3], a significant portion of about 80% of the clock cycle is consumed by interconnections in some advance systems. As there are a lot of wires to be connected, routing becomes more and more difficult. If this is not considered in early design phrases, like floorplanning, unroutable layouts may be resulted. To avoid unnecessary iteration of the design cycle, modern floorplanners always take interconnections into account.

1.4 Motivations and Contributions

In VLSI system design, it is common that a system is consisted of millions of transistors. A good planning in the early design phase is of vital importance as it sets up a ground work for a good layout.

As the functionality of chips increases, chip designs become more and more complicated and involve a huge number of transistors. Beside functionality, chip designs are expected to meet many other requirements, like timing, power consumption, etc. On the other hand, it is favorable to keep the chip size as small as possible. This makes the design process much more difficult than ever.

In the deep submicron era, the number of transistors and interconnections are growing rapidly. The wires are becoming longer and denser. More routing space is needed to ensure design convergence. Bus is a collection of wires to carry a set of signals among different modules. As the complexity of chip design increases, bus routing becomes more and more important. If we do not carefully plan the routes of the buses and reserve sufficient space for them in the layout, there will be a high chance to have a lot of unroutable buses. In order to ease bus routing and avoid unnecessary iteration in the design cycle, we incorporate bus planning in the early designing phase. This is our motivation to solve the bus-driven floorplanning problem.

Our research focused on bus-driven floorplanning, in both 2D and 3D chip design. We have reviewed literatures on floorplanning, which include different floorplan representations and bus planning methods. We used the sequence pair (SP) representation and the transitive closure graph (TCG) representation for 2D and 3D floorplanning respectively.

For 2D floorplanning, we made use of the characteristics of SP and proposed a novel algorithm [23] to solve the bus-driven floorplanning problem, allowing buses with bendings. Given a SP, the topological relationships between the blocks can be found. We have proposed a method to check if buses can be placed in a specific floorplan by studying the relative positions between the blocks as represented by a SP. Simulated annealing was used to find a good solution. We have compared our work with [1], and significant improvement were made.

3D chips are useful in reducing interconnect lengths. However, there is not much previous work done in 3D floorplanning. In this thesis, we have proposed a 3D floorplan representation called *Layered Transitive Closure Graph (LTCG)*. It is based on the Transitive Closure Graph (TCG) representation for non-slicing floorplans, together with some layer information. We proposed a method to align blocks of the same bus on different layers, by adding edges into the LTCG. A floorplanner is implemented using the LTCG representation. Experimental results have shown that LTCG is a promising representation for 3D floorplans and can handle bus planning in 3D floorplans effectively.

1.5 Organization of the Thesis

The rest of this thesis is organized as follows. After giving a brief introduction to the background information in this chapter, a literature review on different 2D floorplan representations will be given in Chapter 2. After that, a literature review on 3D floorplan representations will be given in Chapter 3. In Chapter 4, a literature review on previous approaches to solve the bus-driven floorplanning problem will be presented. Our proposed algorithm to solve the multi-bend bus-driven floorplanning problem in 2D floorplan will be presented in Chapter 5, followed by our proposed representation for 3D floorplans and our approach to perform bus-driven floorplaning for 3D chips in Chapter 6. Finally, a conclusion will be given in Chapter 7.

Chapter 2

Literature Review on 2D Floorplan Representations

2.1 Types of Floorplans

Floorplans can be classified into three main categories: slicing [24][25][26], non-slicing [12][20][27][28][11][19][29], and mosaic [18][30][13][31] as shown in Figure 2.1.

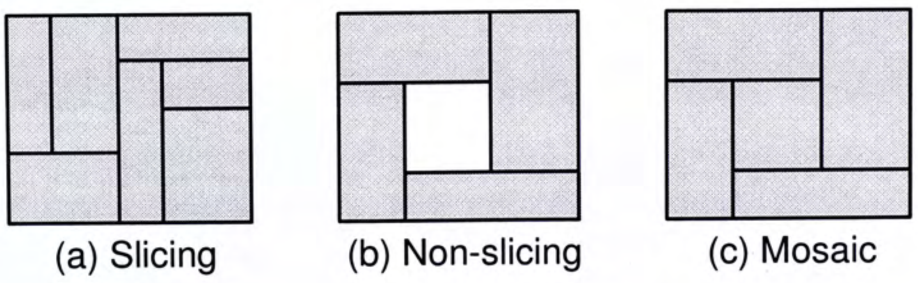


Figure 2.1: Examples of the Three Main Kinds of Floorplans.

A *slicing* structure can be obtained by recursively dividing a rectangle into smaller rectangles using a horizontal or a vertical cut. An example is shown in Figure 2.1(a). A widely adopted slicing floorplan representation is proposed by Wong and Liu in 1986 [25], which is called *normalized Polish expression.* One of the advantages of the slicing structure is that the solution space is smaller, implying faster runtime for some search-based floorplanning algorithms. Solution space refer to how many different solutions can one representation represent. However, the representation is not general enough, as most of the real designs are not in slicing structure.

A *non-slicing* floorplan is a floorplan that is not necessarily slicing (Figure 2.1(b)). It is the most general kind of floorplans. Much work has been done on non-slicing floorplan representation recently, e.g., *sequence pair* [11], *BSG* [29], *O-Tree* [20], *B*-Tree* [12], and *TCG* [32].

Mosaic floorplan is first proposed in 2000 [18], to represent a new class of packing structure. Mosaic floorplan is similar to non-slicing floorplan except that there is no empty room in the floorplan (Figure 2.1(c)). Each module corner is formed by a T-junction (no +-junction), except those at the four corners of the floorplan. Besides, the non-crossing segment of a T-junction can slide along the crossing segment to represent the same floorplan as shown in Figure 2.2.

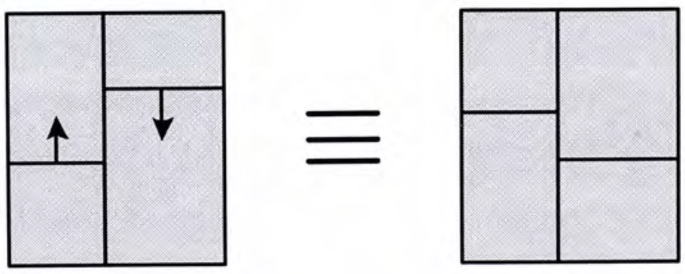


Figure 2.2: One of the Properties of Mosaic Floorplan.

According to [13], the categories of floorplans can be summarized as in Figure 2.3, where slicing floorplan is a proper subset of mosaic floorplan, and mosaic floorplan is a proper subset of general(non-slicing) floorplan.

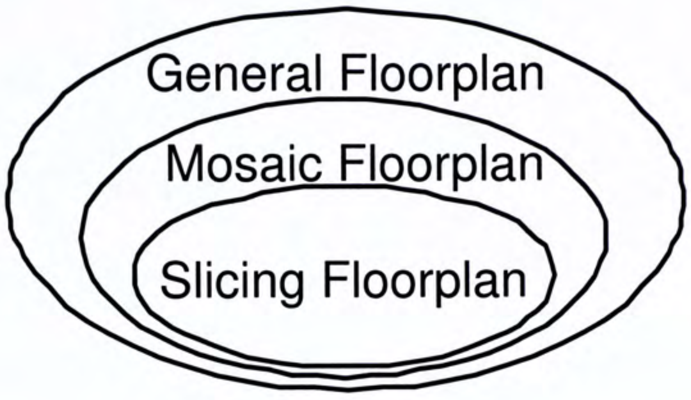


Figure 2.3: Floorplans Categories.

2.2 Floorplan Representations

A good floorplan representation should have the following qualities: small solution space, quick floorplan realization procedure, and being P-admissible.

The notion of *P-admissible* is first proposed by Murata et al. in [11]. For a representation to be P-admissible, it has to satisfy the following four requirements:

1. The solution space is finite,
2. Every solution is feasible,
3. Evaluation for each solution is possible in polynomial time and so is the realization of the corresponding packing,
4. The packing corresponding to the best evaluated solution in the space coincides with an optimal placement solution.

Different representations for different kinds of floorplans will be discussed in the following sections.

2.2.1 Slicing Floorplan

Normalized Polish Expression

According to Wong and Liu in [25], A slicing structure is a rectangle dissection that can be obtained by recursively cutting rectangle into smaller rectangles. The authors suggested to use an oriented rooted binary tree called *slicing tree* to represent the hierarchical structure of a slicing floorplan. Each internal node of such a slicing tree is labelled by a '*' (corresponds to a vertical cut) or a '+' (corresponds to a horizontal cut), while each leaf is labelled by the module name. An encoding to the tree can be obtained by traversing the slicing tree in a post-order, called a *Polish expression.* A Polish expression is said to be normalized if the Polish expression contains no consecutive '*'s nor '+'s. In Figure 2.4, an example of a slicing tree together with its normalized Polish expression is shown.

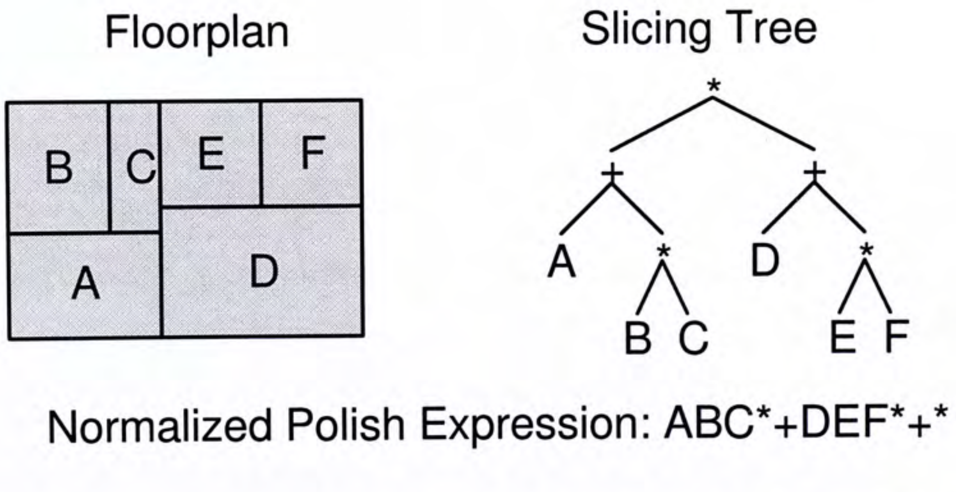


Figure 2.4: An Example of a Slicing Tree and Its Normalized Polish Expression.

In [25], it is shown that there is a one-to-one correspondence between the normalized Polish expressions and the slicing floorplans. The size of the solution space is $O(n!2^{3n-3}/n^{1.5})$[13] where n is the number of modules. A slicing floorplan can be realized from a normalized Polish expression in $O(n)$ time. The representation is P-admissible. Normalized Polish expression is a widely adopted, elegant representation for slicing structures.

2.2.2 Non-slicing Floorplan

Sequence Pair (SP)

Sequence Pair (SP) was first proposed in 1995 by Murata et al [11]. In the representation, two sequences (Γ_+, Γ_-) are used to represent a floorplan. For example, ($ABDECF$, $CBFADE$) is a sequence pair of the set of modules $\{A$, B, C, D, E, $F\}$. The relationship between every two blocks is governed by the following rules:

- If two blocks A and B appear in the sequence pair as $(\cdots A \cdots B \cdots$, $\cdots A \cdots B \cdots)$, block B is on the right of block A.
- If two blocks A and B appear in the sequence pair as $(\cdots A \cdots B \cdots$, $\cdots B \cdots A \cdots)$, block B is below block A.

To realize a floorplan from a sequence pair representation, a pair of graphs, the *horizontal constraint graph* G_H and the *vertical constraint graph* G_V can be constructed. Each constraint graph has a source s and a sink t to denote the floorplan boundaries. In G_H, the source and the sink correspond to the leftmost and the rightmost boundaries of the floorplan respectively, while in G_V, the source and the sink correspond to the bottommost and uppermost boundaries of the floorplan respectively. The constraint graphs are vertex-weighted, and the set of vertices V is $\{s\} \cup \{t\} \cup \{v_1, v_2, \cdots, v_n\}$, where n is the number of modules, and each v_i corresponds to a module. The vertex-weight is zero for s and t in both graphs, and is the width(height) of the corresponding module in $G_H(G_V)$. The constraint graphs can be constructed as follows (Figure 2.5):

- If block A is on the left of block B, add an edge (A, B) in G_H.
- If block A is below block B, add an edge (A, B) in G_V.

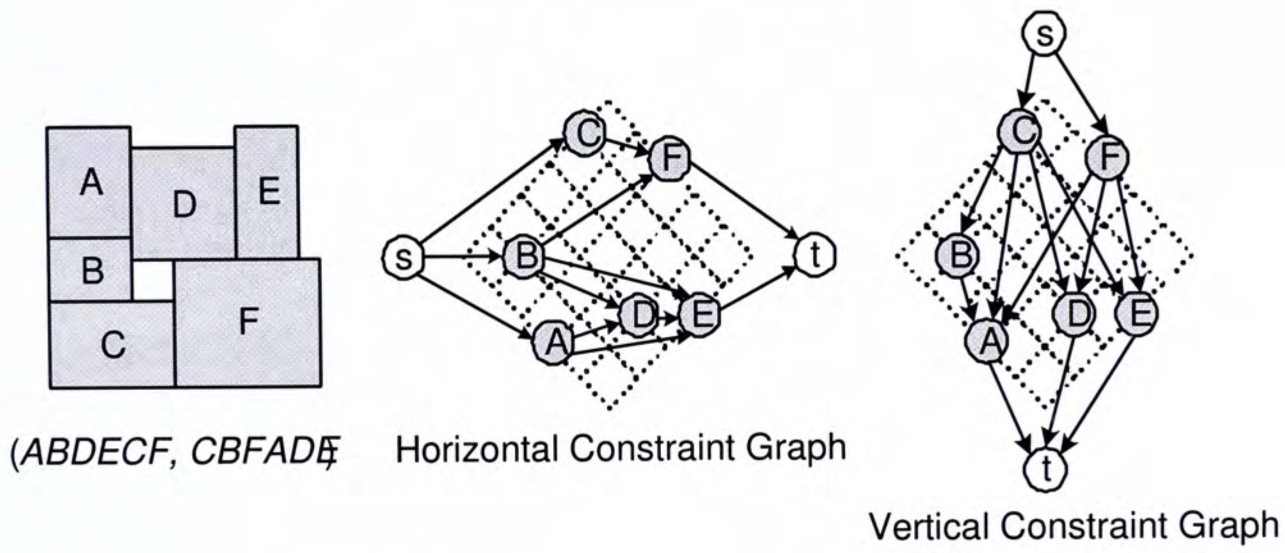


Figure 2.5: Constraint Graphs for the Sequence Pair $(ABDECF, CBFADE)$.

Sequence pair is a P-admissible representation. The time complexity of realization of a floorplan from a SP is $O(n^2)$ according to [11], where n is the number of modules, and is improved to $O(nloglogn)$ in [33]. The size of the solution space of SP is $O((n!)^2)$.

Bounded-Sliceline Grid (BSG)

BSG refers to *Bounded-Sliceline Grid.* It is a non-slicing floorplan representation proposed in 1996 by Nakatake et al. in [19] based on the topological relationships between blocks. A meta-grid is defined on a plane without any physical dimension. Different segments in the grid create rooms to place different blocks. The unit segment on the (x, y)-coordinate system is defined by:

$$H_{i,j} = \{(x, y) \mid i-1 < x < i+1, y = j\}$$
$$V_{i,j} = \{(x, y) \mid x = i, j-1 < y < j+1\}$$

A BSG consists of a set U_{BSG} of the unit segments as defined above. An example is shown in Figure 2.6.

$$\begin{aligned} U_{BSG} = \; & \{V_{i,j} \mid i, j : integers, i+j : even\} \cup \\ & \{H_{i,j} \mid i, j : integers, i+j : odd\} \end{aligned}$$

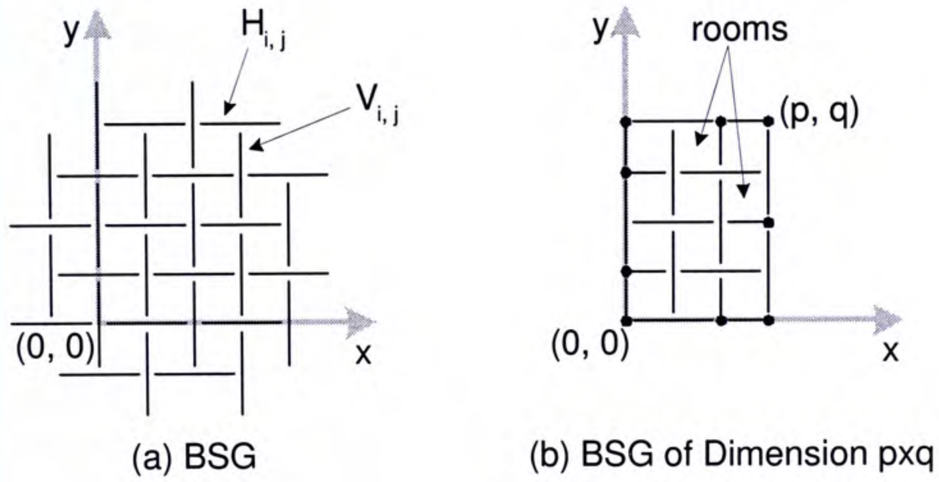


Figure 2.6: (a) An Example of a BSG. (b) A Domain BSG$_{p \times q}$

A pair of graphs, the horizontal unit adjacency graph G_H and the vertical unit adjacency graph G_V, can be constructed to realize the floorplan. In G_H, each vertex corresponds to a horizontal segment. Edges are added between adjacent segments and thus, each edge crosses one room. If an edge e crosses a non-empty room where block A is placed, the weight of e will be the width of block A. If e crosses an empty room, the weight of e will be 0. G_V can be built in a similar fashion. With the constructed graphs, the layout of the floorplan can be found by performing longest path search for every block. An example of representing a floorplan using BSG is showin in Figure 2.7.

According to [19], BSG is beneficial when packing blocks into a chip of non-rectangular shape. BSG is a P-admissible representation. To realize a floorplan from its BSG representation, the time complexity is $O(n^2)$. The size of the solution space of BSG is $O(n^2!/(n^2 - n)!)$.

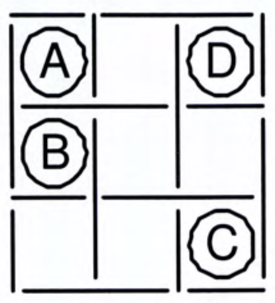
O-Tree

In 1999, Guo et al. proposed an *O-tree* representation for non-slicing floorplan in [20]. They defined an *admissible placement* as a compacted placement where all blocks can neither move down nor move left. O-tree is devised to represent admissible placement.

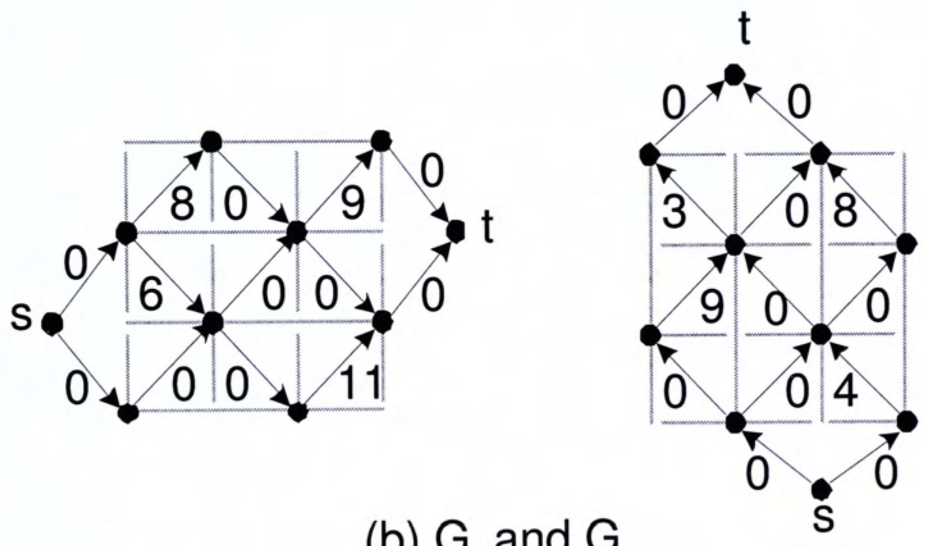
Given a floorplan, two different ordered trees can be built, one with the root corresponding to the left boundary of the floorplan and one with the root corresponding to the bottom boundary of the floorplan. Given an O-tree, its orthogonal correspondent can be built. An example of an admissible placement and its corresponding O-tree is shown in Figure 2.8. The root node in the figure corresponds to the left boundary of the floorplan.

The authors proposed to encode the rooted ordered tree into two sequences (T, π). The sequence T indicates the structure of the tree: a '0' represents a descending edge and a '1' represents an ascending edge. The sequence π is a sequence of module labels obtained by performing a depth-first search. Thus, the floorplan in Figure 2.8 is represented by (00110100011011, $ADBCEGF$).

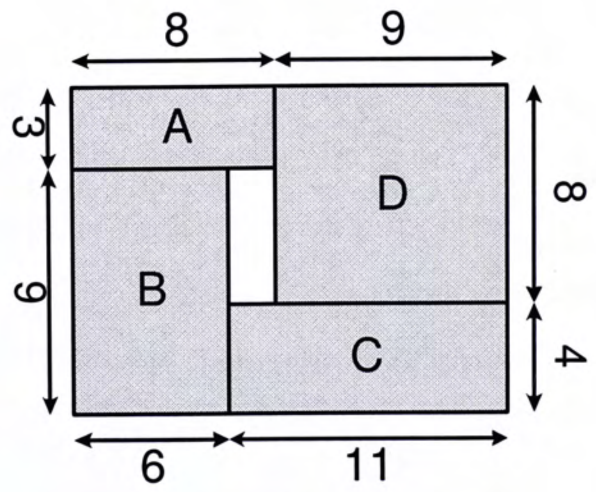
This representation is not P-admissible. The size of the solution space of



(a) Assignment of Rooms



(b) G_H and G_V



(c) The Resultant Packing

Figure 2.7: Representing a Floorplan Using BSG.

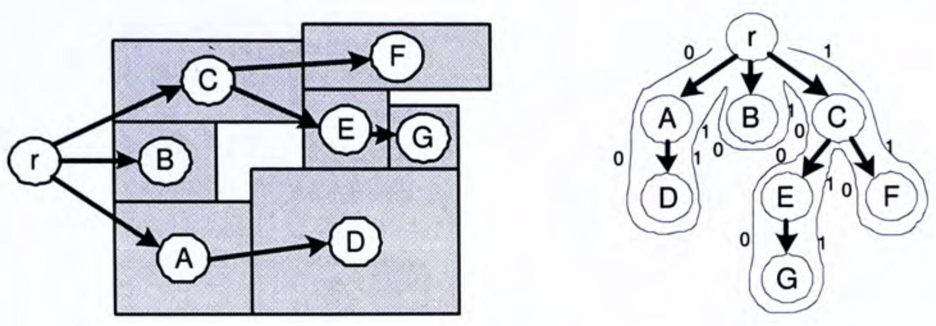


Figure 2.8: An Admissible Placement and Its Horizontal Constraint Graph.

O-tree is $O(n!2^{2n-2}/n^{1.5})$, and the runtime to transform an O-tree to its packing is linear, i.e., $O(n)$.

B*-Tree

B*-tree[12] is proposed in 2000 by Chang et al. which is similar to O-tree, with some modifications and enhancements.

Each admissible placement has a corresponding B*-tree T. Each node in T corresponds to a module. The root node of T corresponds to the module at the bottom-left corner of the floorplan. Let R be the set of modules on the right-hand side of and adjacent to a block x. The left child of the node x in T is the lowest unvisited block in R. Similarly, let U be the set of modules above and adjacent to x, the right child of x in T is the leftmost unvisited block in U. According to [12], there is a one-to-one correspondence between admissible placement and B*-tree. An example of a placement and its B*-tree representation is shown in Figure 2.9.

B*-tree is advantageous over O-tree, as B*-tree is a binary tree and it can be implemented easily with a static data structure such that node searching

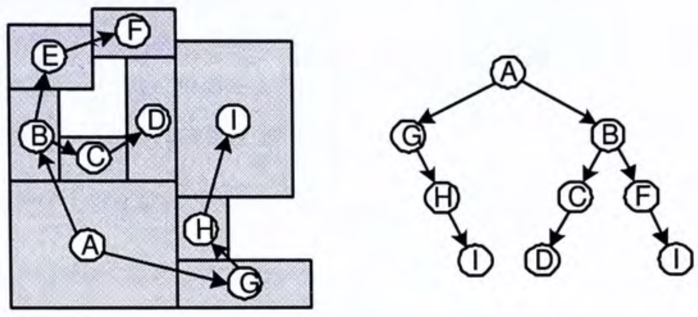


Figure 2.9: A Floorplan and Its Corresponding B*-Tree Representation.

and insertion can be done in constant time, i.e., $O(1)$. Similar to O-tree, B*-tree is not P-admissible. The size of its solution space is $O(n!2^{2n-2}/n^{1.5})$, and floorplan realization takes $O(n)$ time.

Transitive Closure Graph (TCG)

In 2001, *Transitive Closure Graph* (TCG) is proposed by Lin and Chang in [32] to represent non-slicing floorplan. A TCG is a pair of directed acyclic graph, the horizontal transitive closure graph C_h and the vertical transitive closure graph C_v. The authors defined the *transitive closure* G' in of a directed acyclic graph $G = (V, E)$ as follows: $G' = (V, E')$, where $E' = \{(n_i, n_j)$: there is a path from node n_i to node n_j in $G\}$.

The authors made use of the topological relationships between blocks to represent a floorplan. For two non-overlapping modules b_i and b_j, they must bear one of the following three relationships: (1) horizontal relation, (2) vertical relation, or (3) diagonal relation. The first two relationships are easy to understand: the two modules are overlapped in one dimension but not the other. For the third one, b_i is said to be *diagonally related* to b_j if the projections of the two modules do not overlap in either dimension. For simplicity, a diagonal relationship will be treated as a horizontal one, unless there exists a chain of vertical relations (e.g. if A is diagonally related to C, but A is above B and B is above C, then A must be above C). For two blocks bearing a horizontal relationship, an edge will be added in C_h between the nodes representing the two blocks, while edges in C_v will correspond to vertical relationships. The graphs are vertex-weighted, where the weights correspond to the widths (heights) of the blocks. Figure 2.10 shows an example of representing a floorplan using TCG.

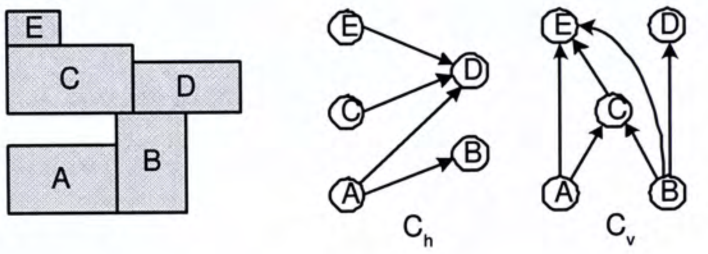


Figure 2.10: A Floorplan and Its Corresponding TCG Representation.

Realizing a floorplan from its TCG representation is easy. It can be done by performing a longest path search on the two constraint graphs. It is claimed that TGC has several advantages over some published work: TGC is P-admissible, TGC does not need sequence encoding, cost can be evaluated directly basing on the representation, and geometric relationship is transparent to its operations, etc. The size of the solution space of TGC is $O((n!)^2)$ and floorplan realization can be done in $O(n^2)$ time.

2.2.3 Mosaic Floorplan

Corner Block List (CBL)

Corner Block List (CBL) is a topological representation for mosaic floorplan. It is first proposed by Hong et al. in 2000 [18]. A *corner block* is the upper-rightmost block in a floorplan. A CBL is a three tuple (S, L, T), that can be obtained by repeatedly deleting the corner block of the floorplan.

The sequence S is a sequence of block names. It records the order of the blocks being deleted. L is a list of orientations. The orientation of a block is defined according to the T-junction at its bottom left corner. There are two kinds of orientations: a 'T' rotated by 90 degrees anticlockwise ($\vdash$) or by 180 degrees ($\perp$). In the former case, a '0' will be recorded in L and in the latter case, a '1' will be recorded. The list T records the number of T-junctions on the left or bottom boundary of the corner blocks. The number of consecutive '1's in T corresponds to the number of T-junctions on the left or bottom boundary of a corner block. A '0' is added to separate this information for different blocks. The orientation and the T-junction information of the last block will not be recorded as there is only one block left at the end of the deletion process. An example is shown in Figure 2.11 to illustrate the process of obtaining the CBL from a packing.

Floorplan realization for CBL can be done in a similar fashion as in CBL construction. It can be done by checking the orientation of the corner block, and determining whether the horizontal segment or the vertical segment of the corner block should be pushed to make a room. According to [13], the size of the solution space of CBL is $O(n!2^{3n}/n^{1.5})$. The computation complexity to convert a CBL to a floorplan is $O(n)$. However, CBL is not a P-admissible representation.

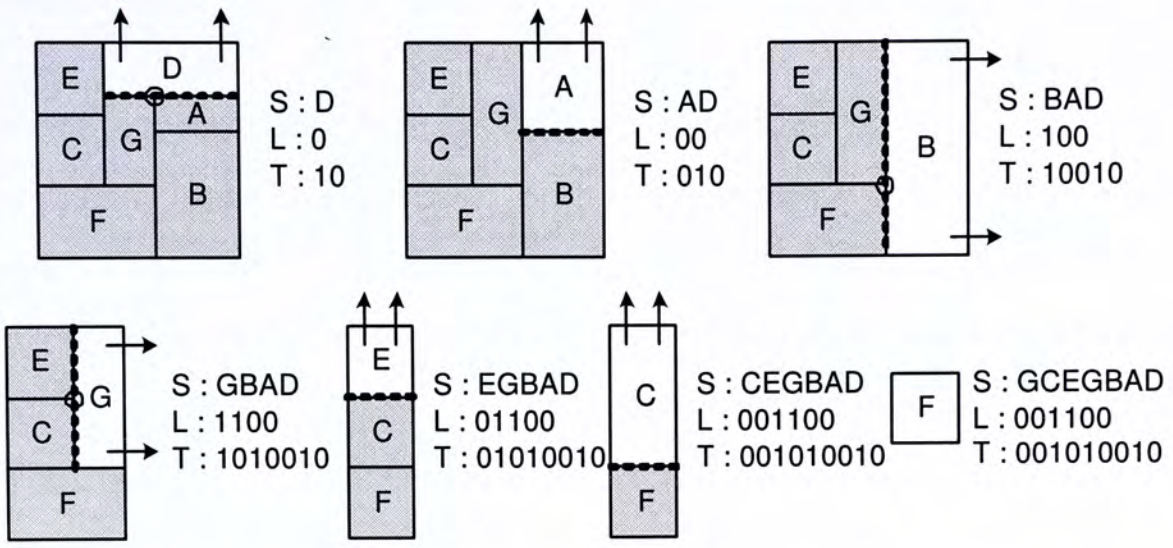


Figure 2.11: Constructing a CBL from A Floorplan.

Twins Binary Trees (TBT)

Twins Binary Tree (TBT) is first proposed to be used as a floorplan representation by Yao et al. in 2001 [30]. It is proved that there exists an one-to-one mapping between TBT and mosaic floorplan. According to [30], the set of twins binary trees $TBT_n \subseteq Tree_n \times Tree_n$ is defined as followed:

$$TBT_n = \{(b_1, b_2) | b_1, b_2 \in Tree_n \text{ and } \Theta(b_1) = \Theta^c(b_2)\}$$

where $Tree_n$ is the set of binary trees with n nodes, and $\Theta(b)$ is the labelling of a binary tree.

The labelling of a tree can be obtained by carrying out an in-order walk on the tree. Beginning with an empty sequence, a '0' is added to the sequence if a node with no left child is being visited, and a '1' is added to the sequence if a node with no right child is being visited. The first '0' and the last '1' in the sequence are omitted. The complement $\Theta^c(t_1)$ of $\Theta(t_1)$ is obtained by interchanging the '0' and '1' bits.

Given a mosaic floorplan, its TBT representation (t_1, t_2) can be obtained by traversing along the slicelines. Both trees contain n nodes, where n is the number of modules in the floorplan. The root of t_1 is the bottom-left corner block of the floorplan. The tree t_1 is built by connecting the bottom-left corners of all the blocks. The left tree edge of a node represents the vertical sliceline, and the right tree edgerepresents the horizontal sliceline. t_2 is built similarly: the root is the upper-right corner blcok of the floorplan, and the tree is built by connecting the upper-right corners of all the blocks. The left tree edgeof a node represents the horizontal slicelnie, and the right tree edge of a node represents the vertical sliceline. It is proved that the pair of trees constructed in this way must be twin binary to each other. An example of a floorplan and its TBT representation is shown in Figure 2.12. The solution space of TBT is $O(n!2^{3n}/n^{1.5})$.

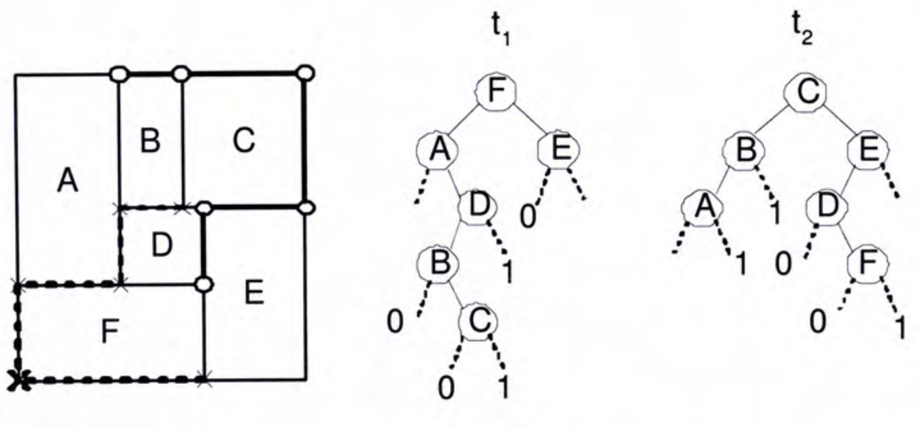


Figure 2.12: A Non-Slicing Floorplan and Its TBT Representation.

Twins Binary Sequences (TBS)

In 2002, Young et al. proposed a *Twins Binary Sequences* (TBS) representation for mosaic floorplan in [13] basing on TBT. The idea of using TBT to represent mosaic floorplans was proposed in [30] but the exact modelling was not mentioned. For example, it is not known that how the nodes in the TBT should be labelled so that it corresponds to a feasible floorplan. In view of this, the authors in [13] proposed a TBS representation to cope with the problems. The authors proposed to use a 4-tuple $s = (\pi, \alpha, \beta, \beta')$ to represent a floorplan. We call s the TBS representation of a floorplan.

π is the in-order traversals of the twin binary trees, and α is the labelling of them. The authors claimed that a pair of twins binary trees will correspond to a feasible packing if and only if their in-order traversals are the same. However, these two pieces of information solely are not enough to represent a floorplan uniquely. Thus, two more bit sequences β and β' are needed. These two sequences record the structural information of the trees: a bit '0' represents the root of a tree and a node that is the right child of its parent, and a bit '1' represents a node that is the left child of its parent. β is used to represent the directional information of t_1, where β' is used to represent the directional information of t_2. An example is shown in Figure 2.13.

To make TBS more general, the authors proposed to include in the input some dummy zero-area blocks. They have proved that a tight bound of $\theta(n)$ dummy blocks are needed to obtain general non-slicing floorplan from mosaic floorplan.

Realizing a floorplan from its TBS representation is very efficient according to [13]. It can be done by scanning the sequences only once from right to left.

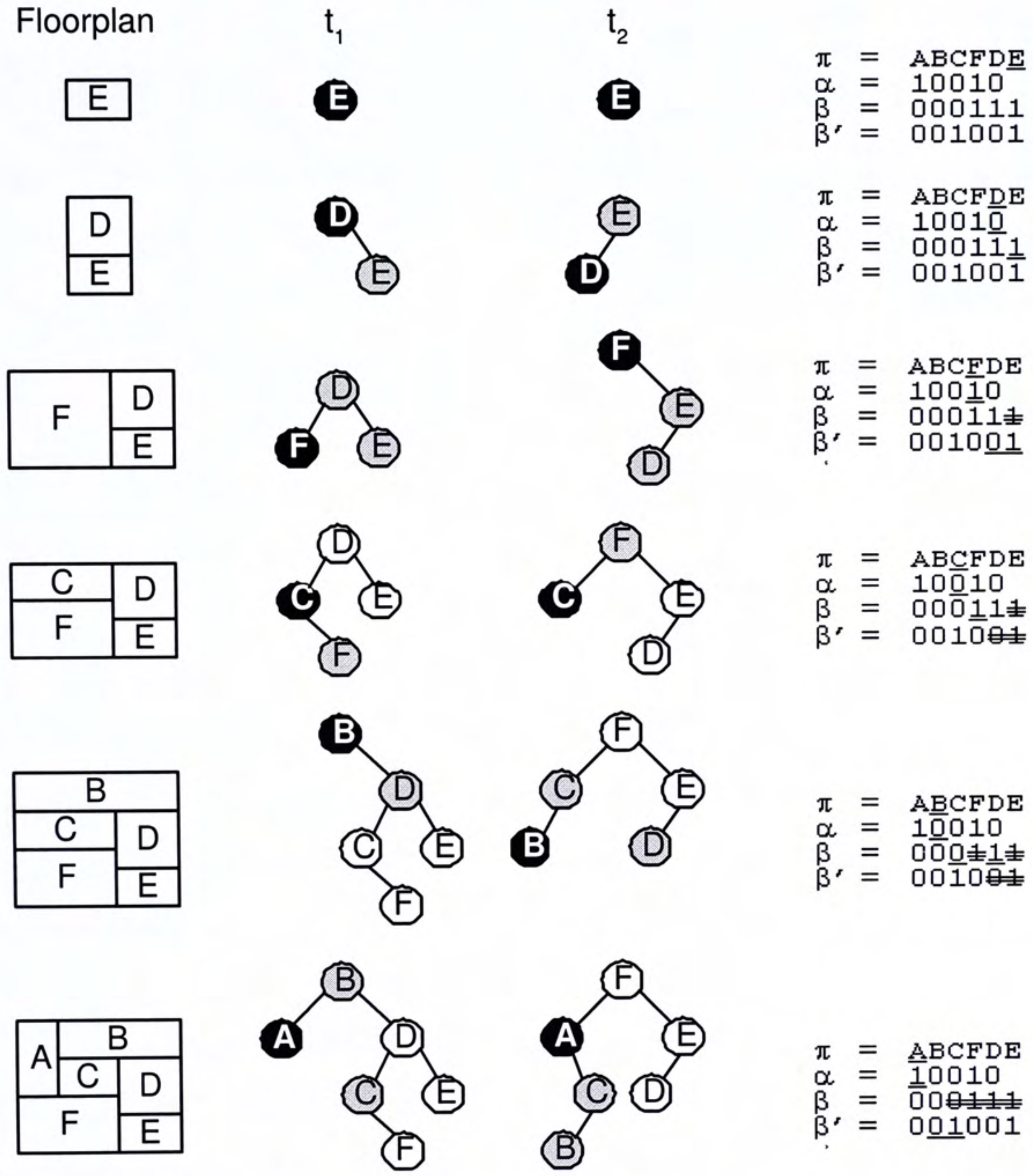


Figure 2.13: A Floorplan Realization Example using TBS.

It is also proved that there is a one-to-one mapping between TBS and TBT, and thus a one-to-one mapping between TBS and mosaic floorplan. The size of the solution space of TBS is the same as that of TBT, $O(n!2^{3n}/n^{1.5})$.

2.3 Summary

In this chapter, different types of floorplan are introduced, which are slicing, non-slicing, and mosaic floorplan. Slicing floorplans are obtained by recursively dividing a rectangle into smaller rectangles. Though the solution space is small, slicing floorplans are not general enough, as most floorplans are not slicing practically. Mosaic floorplans are not necessarily obtained by dividing rectangles and thus are more general but they contain no empty space. Non-slicing floorplan is the most general one.

Current state-of-the-art representations of each type of floorplan are presented. Table 2.1 is a table summarizing the characteristics of these representations. For slicing floorplans, the most popular representation used is the normalized Polish Expression [25]. The representation is simple and elegant, and floorplan realization can be done in linear time.

For non-slicing floorplans, there are several representations such as sequence pair (SP), bounded-sliceline grid (BSG), O-Tree, B*-Tree, and transitive closure graph (TCG). The sizes of their solution spaces and the floorplan realization runtimes are different. SP, BSG, and TCG are P-admissible where O-Tree and B*-Tree are not.

Mosaic floorplans can be represented using corner block list (CBL), twins binary tree (TBT), or twins binary sequences (TBS). The solution space of

CBL is small, but not all CBL corresponds to a floorplan. In TBT and TBS, the solution is one-to-one mapped to the representation and the realization process can be done in linear time.

Representation	Size of Solution Space	Time Complexity of Floorplan Realization
Normalized PE	$O(n!2^{5n-3}/n^{1.5})$	$O(n)$
SP	$O((n!)^2)$	$O(nloglogn)$
BSG	$O(n^2!/(n^2-n)!)$	$O(n^2)$
O-Tree	$O(n!2^{2n-2}/n^{1.5})$	$O(n)$
B*-Tree	$O(n!2^{2n-2}/n^{1.5})$	$O(n)$
TCG	$O((n!)^2)$	$O(n^2)$
CBL	$O(n!2^{3n})$	$O(n)$
TBT	$O(n!2^{3n}/n^{1.5})$	$O(n)$
TBS	$O(n!2^{3n}/n^{1.5})$	$O(n)$

Table 2.1: Comparison between Different Kinds of Floorplan Representations.

Chapter 3

Literature Review on 3D Floorplan Representations

3.1 Introduction

As the VLSI design complexity increases, both the number of blocks and the number of interconnects involved have increased dramatically. Interconnect awareness in every step of the design cycle has become a major concern as technology advances into the deep submicron era. In view of this, 3D chip is proposed. Interconnect lengths can be reduced greatly in 3D chips and thus making it easier to meet the timing requirements and to reduce the interconnect cost. Unlike the traditional packing problem of 3D blocks, there are several layers available for placing modules in 3D floorplanning. Thus, 3D floorplans are also known as multi-layer floorplans.

Though 3D chips are advantageous in solving the interconnect problem, there are still a lot of design challenges and there are not yet enough EDA tools to assist 3D chip design. In this chapter, some previous work on floorplanning for 3D chips will be discussed.

3.2 Problem Formulation

The formal definition of the floorplanning problem for 3D design are given as followed:

Definition 3.1 The input is a set of n modules $\{M_1, M_2, \cdots, M_n\}$ and a value K that represents the number of layers, where each module M_i is associated with an area A_i, together with two aspect ratio bounds r_i and s_i such that $r_i \leq h_i/w_i \leq s_i$, where h_i and w_i are the height and the width of module i respectively. The output of the problem is a packing of the set of modules, i.e., the x- and y-coordinates and the dimensions (h_i, w_i) of modules i, and the layer l_i, where $1 \leq l_i \leq K$, on which module i lies. There should be no overlapping between modules in each layer, and the circuit performance should be optimized.

3.3 Previous Work

Several researchers have worked on floorplanning for 3D chips recently. They have proposed different representations for 3D floorplans. Their work will be reviewed in this section.

Slicing Tree

In 2004, the authors of [4] proposed a slicing structure representation for multi-layer floorplans. In 2D floorplan representation, a floorplan is said to be a *slicing structure* if it can be obtained by recursively dividing a rectangle into two by vertical or horizontal lines. The authors extend this idea into three dimensions, and adopted the Normalized Polish Expressions to represent multi-layer floorplans.

Similar to Normalized Polish Expressions for 2D floorplans, a slicing tree is constructed for multi-layer floorplans. There are three kinds of internal nodes, 'H', 'V', and 'Z', representing horizontal, vertical, and lateral cuts respectively, while each leaf is labelled by a module name.

To realize the floorplan, the slicing tree has to be broken down. Each layer is represented by a slicing sub-tree. This is done by removing all the 'Z' nodes in the tree, leaving behind those 'V' and 'H' nodes only.

Given a slicing tree, we will construct the slicing sub-tree for each layer one by one from the top to the bottom. At each layer, the 'Z' node is replaced by its left child, and the right sub-tree is put to the lower layer. To put a sub-tree to the lower layer, it is checked whether the lower layer is empty first. If so, the sub-tree becomes the slicing sub-tree of that layer. Otherwise, a new root node is created to join the current slicing sub-tree and the newly added sub-tree and the label on the new root is either 'V' or 'H' depending on the lowest common ancestor of these two subtrees in the original slicing tree. An example showing a multi-layer floorplan and its slicing tree representation is illustrated in Figure 3.1.

An Array of 2D Representations

To make things easy and strict forward, some researchers have proposed to use an array of 2D representations to represent multi-layer floorplans [34][35]. In [34], the authors proposed to use an array of Base Slice-line Grid (BSG) to represent a multi-layer floorplan, where each BSG represents the 2D floorplan on each layer. In [35], the same approach is used, but sequence pair is selected as the 2D floorplan representation.

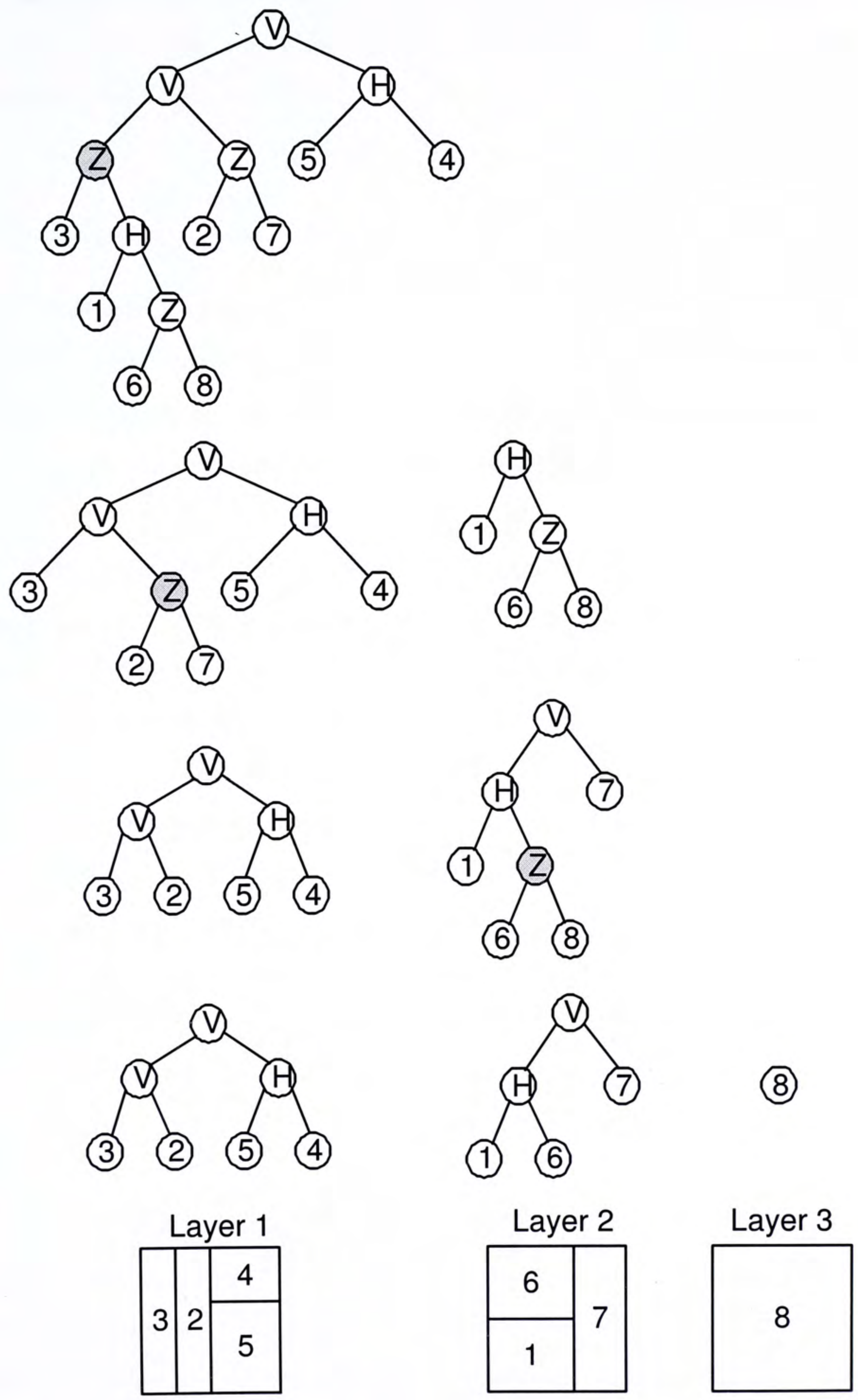


Figure 3.1: (a) A 3D Slicing Tree. (b) The 2D Slicing Tree and the Floorplans of Each Layer.

This kind of representations is strict forward and easy to understand. However, the relationships between blocks in different layers can not be reflected by the representation solely.

Combined Bucket and 2D Array (CBA)

To represent multi-layer floorplans, we can use a 2D representation to represent each layer. However this is not good as the relationships between blocks in different layers are not stored. In view of this, the authors of [36] proposed a multi-layer representation called Combined Bucket and 2D Array (CBA).

CBA is consisted of two parts, a 2D representation to represent each layer, and a bucket structure to store the relationships between blocks in different layers. In [36], TCG is selected as the 2D representation but in fact, any 2D floorplan representation like Sequence Pair or Corner Block List can be used.

A bucket represents a rectangular region on the x-y plane. It stores the relationships between blocks in different layers. For each bucket, indexes of the blocks that intersect with that bucket are stored. Besides, for each block, the indexes of the buckets that intersect with that block are also recorded. Thus, if two block i and j, locating in different layers, intersect with the same bucket k, it is likely that they are placed close to each other. In Figure 3.2, a multi-layer floorplan and its CBA representation is shown.

Simulated annealing is used in [36] to search for a good floorplan. The authors have proposed different kinds of moves. Apart from some intra-layer moves like 'rotation', 'swap', 'reverse', and 'move', the authors suggested three more inter-layer moves namely 'interlayer swap', 'z-neighbor swap', and 'z-neighbor move'. The first one means swapping two blocks in different layers.

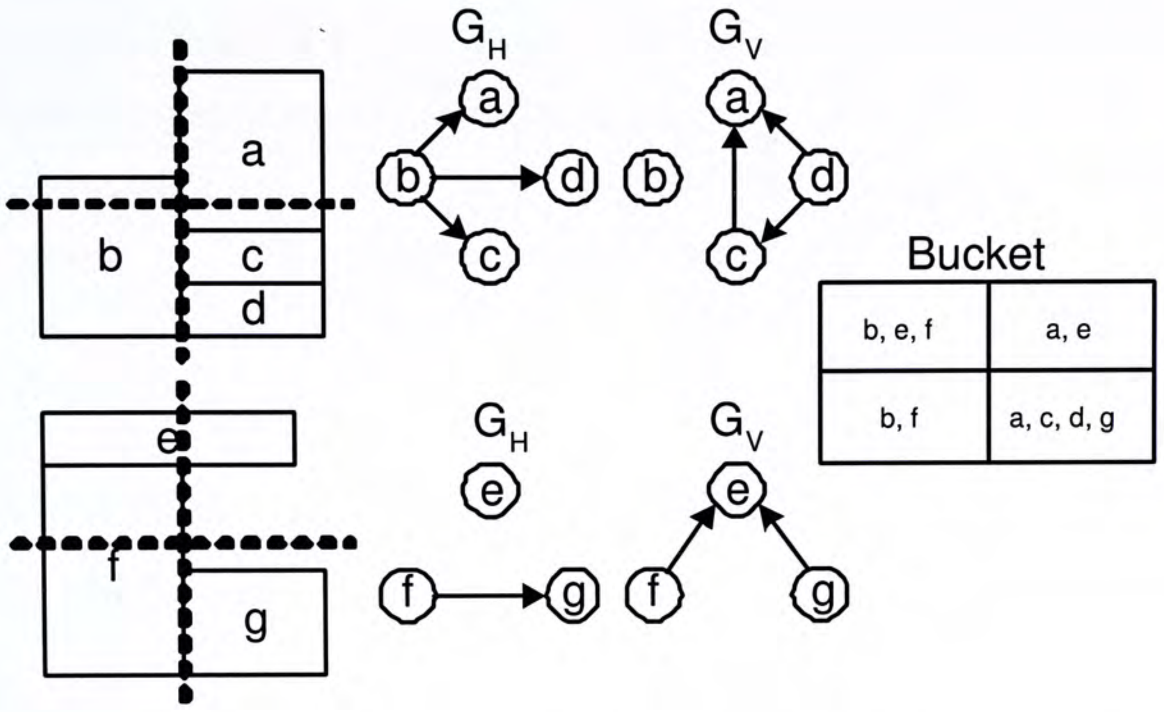


Figure 3.2: A Floorplan Represented by CBA.

The second one means swapping two blocks in different layers, but they must be close to each other. The third one means moving a block to another layer, and the destination must be close to its original position. Experimental results showed that the performance of [36] is better than that of [35].

3.4 Summary

In this chapter, the mutli-layer floorplanning problem is defined. It is different from the traditional floorplanning problem, as it allows blocks to be placed on more than one layer. Multi-layer floorplan design is beneficial as it can reduce the interconnect cost significantly, making the routing step easier, and making it easier to meet the timing requirements.

Several previous work on multi-layer floorplan representation is reviewed.

In [4], a slicing tree representation is proposed and in [34] [35], an array of 2D floorplan representations is proposed. However, for both of them, the relationships between blocks in different layers are neglected. Thus, the authors of [36] proposed a Combined Bucket and 2D Array representation to extend the state-of-the-art 2D representations to multi-layer.

Chapter 4

Literature Review on Bus-Driven Floorplanning

4.1 Problem Formulation

Bus-Driven Floorplanning problem is a floorplanning problem with bus planning taken into consideration. Bus is a collection of interconnections between a set of modules. The problem of bus-driven floorplanning (BDF) can be defined as follows [1]:

Definition 4.1 Bus-Driven Floorplanning (BDF)
Given the following:

1. A set of n blocks $B = \{b_0, b_1, ..., b_{n-1}\}$, where each block b_i is associated with a width w_i and a height h_i, where w_i, $h_i \in \mathbf{R}^+$.

2. A set of m buses $U = \{u_0, u_1, ..., u_{m-1}\}$, where each bus u_i has a width t_i, $t_i \in \mathbf{R}^+$, and goes through a set of blocks B_i, $B_i \subseteq B$.

Our task is to decide the position of each block and the route of each bus, such that each bus u_i can go through all its blocks. There should be no overlapping between any two blocks. The goal is to minimize the chip area and the total bus area.

Some recent approaches used to solve the bus-driven floorplanning problem will be discussed in the coming sections.

4.2 Previous Work

Many algorithms have been proposed to enforce different kinds of placement constraints in floorplan design. For example, the authors in [37][38][39][40][41] had considered alignment and abutment constraints in floorplan design. In [1][42], the bus-driven floorplanning problem is addressed. These approaches will be discussed in details in the following sections.

4.2.1 Abutment Constraint

The authors of [37] enforce abutment constraint in floorplanning in order to handle rectilinear block placement. The sequence pair representation is adopted. To take care of rectilinear blocks, the authors proposed to partition each rectilinear block into a set of rectangular sub-blocks. Each block is partitioned in one direction only, and all neighboring sub-blocks are orthogonally aligned (Figure 4.1). Some rectilinear blocks with complicated shape may need to be partitioned into L-shaped sub-blocks, and then into rectangular shapes. In order to employ the approach proposed, the partitioning has to be done in such a way that the neighboring sub-blocks can be grouped into a L-shape block. However, some rectilinear blocks cannot be partitioned according to the above requirements. Then, an ϵ-approximation is performed to divide it into two L-shape sub-blocks. An example is illustrated in Figure 4.2.

After partitioning, the sub-blocks have to be abutted to maintain the original rectilinear shape. For example, if a block X is partitioned into three sub-blocks as in Figure 4.1(a)), they have to be abutted horizontally or vertically in the final floorplan in order to get back the original rectilinear shape.

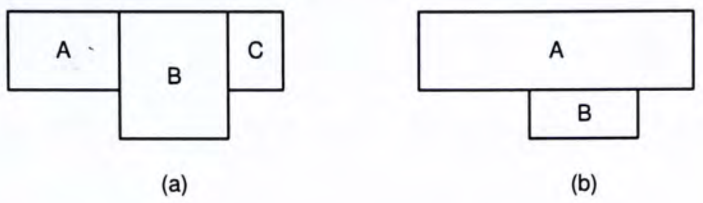


Figure 4.1: (a) A Feasible Partitioning. (b) An Infeasible Partitioning.

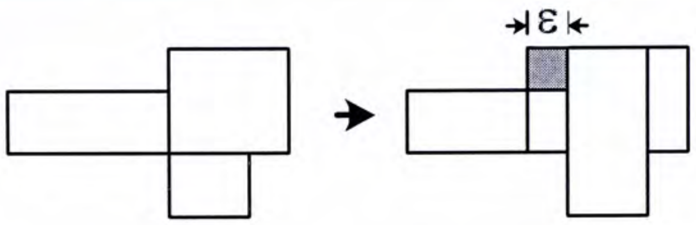


Figure 4.2: A Rectilinear Block That Cannot Be Decomposed into Two L-Shape Sub-Blocks and Its ϵ-Approximation.

There is a key observation: the sub-blocks should maintain their initial relative positions in any feasible placements, e.g., the blocks should appear in the sequence pair as $(...A...B...C..., ...A...B...C...)$, $(...A...B...C..., ...C...B...A...)$, $(...C...B...A..., ...C...B...A...)$, or $(...C...B...A..., ...A...B...C...)$ for the example in Figure 4.1(a). Simulated annealing is used. Infeasible candidate solutions, e.g., the sub-blocks are not abutted, will be penalized in the cost function.

In 2001, the authors of [38] have proposed an algorithm to enforce abutment constraints to blocks in a floorplan. L-shaped and T-shaped blocks are first partitioned into rectangular sub-blocks, and the sub-blocks are then forced to obey the abutment constraints and the rectilinear blocks can thus be placed.

Unlike [37], Corner Block List (CBL) is used to represent a floorplan. The authors have showed that the abutment information of the blocks can be deduced from the CBL representation. Let $HSEG$ be a horizontal segment in a floorplan P, $B_{HSEG} = \{B_1, B_2, \cdots, B_p\}$ denotes the p blocks lying immediately below $HSEG$, and $T_{HSEG} = \{T_1, T_2, \cdots, T_q\}$ denotes the q blocks lying immediately above $HSEG$. If q equals one, every block in B_{HSEG} is lying immediately below the block T_1, implying an abutment information. The case of p equaling one is similar. If both p and q are greater than or equal to two, B_1 will abut with T_1, and B_p will abut with T_q (Figure 4.3).

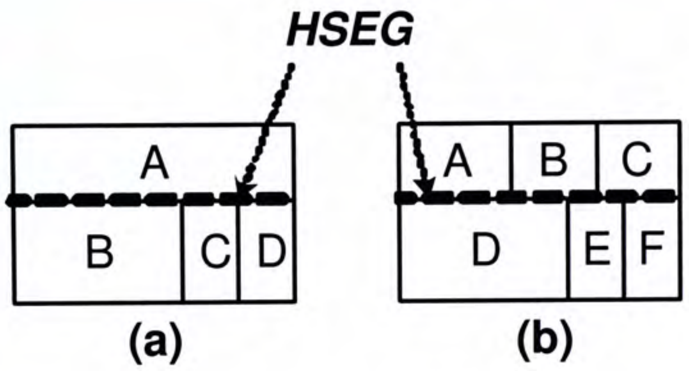


Figure 4.3: (a)Block A is Abutted With Block B, C, and D. (b)Block A is Abutted With Block D, Block C is Abutted With Block F.

To place L-shape or T-shape blocks, they are first partitioned into rectangular sub-blocks. However, enforcing only the abutment constraints to the sub-blocks is not enough. An example is illustrated in Figure 4.4. Thus, the authors introduced the align-abutment constraints, which means the blocks has to be aligned and abutted at the same time. Then, simulated annealing is used to search for a good solution. A penalty will be given to the candidate floorplan solutions in which the align-abutment constraints is violated.

An algorithm to handle arbitrarily shaped rectilinear blocks were proposed in 2004 by Tang et al in [41]. They also used the sequence pair representation.

According to the paper, a rectilinear block is said to be *H-sequential* if

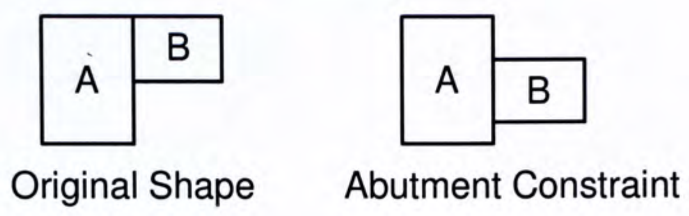


Figure 4.4: Abutment Constraint Alone is Not Enough to Form a L-Shape.

no single vertical line can cut the block into more than two parts. Similarly, a rectilinear block is said to be *V-sequential* if no single horizontal line can cut the block into more than two parts. An example is illustrated in Figure 4.5. A block is said to be *non-sequential* if it is neither H-sequential nor V-sequential (Figure 4.6). If a block is H-sequential, it can be partitioned into a set of horizontally-abutted sub-blocks, the set of sub-blocks are called a H-sequential sequence. For the set of sub-blocks, the relative position of them has to be the same in both sequence of the sequence pair representation. V-sequential sequence can be defined in a similar fashion. An orthogonal link list is proposed to store the information of the rectilinear blocks.

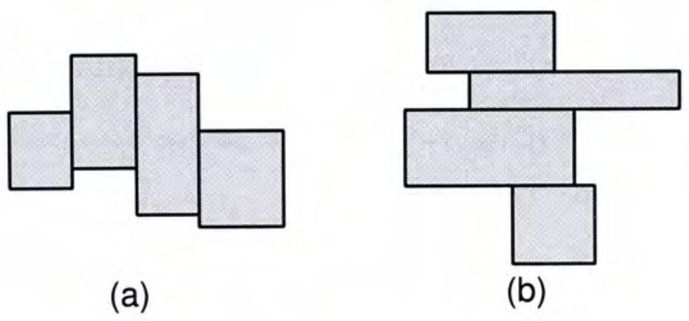


Figure 4.5: (a) A H-Sequential Rectilinear Block. (b) A V-Sequential Rectilinear Block.

Simulated annealing will then be applied to search for a good solution. Experimental results showed that the performance of the approach is promising.

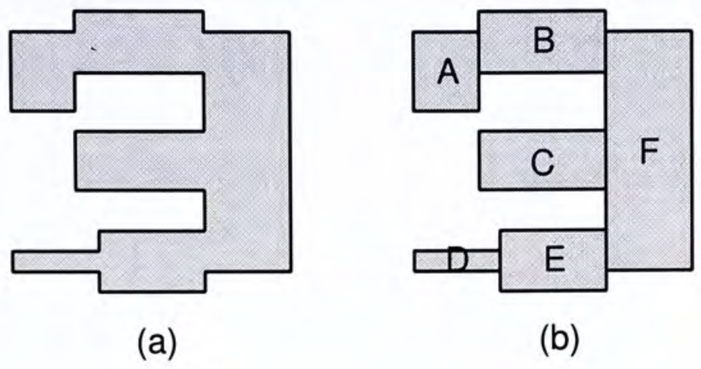


Figure 4.6: (a) A Non-Sequential Rectilinear Block. (b) It is Partitioned into Several Sub-Blocks.

However, the algorithms proposed cannot be applied directly in the bus-driven floorplanning problem, as for a bus to go through a set of blocks, it is not necessary for the blocks to abut with one another. Besides, the order in which a bus goes through its blocks is not known beforehand. Nevertheless, their novel notion of checking relative positions between blocks in a representation is helpful.

4.2.2 Alignment Constraint

In [39], the authors proposed a unified method to handle different kinds of placement constraints, like pre-placed constraint, range constraint, boundary constraint, alignment, abutment, and clustering constraint, etc.

The authors proposed that all the constraints mentioned above can be modelled as a collection of *relative placement constraint* and *absolute placement constraint.* Relative placement constraints are vertical or horizontal distance restriction (a certain range of values) between two modules. For example, $h(A,\ B) = [\alpha,\ \beta]$ means that the horizontal distance between the lower left corners of block A and block B has to be greater than α, but cannot exceed β (Figure 4.7). Absolute placement constraints are similar, except that one of the two modules in the relationship is a boundary of the chip. The left, right, bottom, and top boundary of a chip are denoted by LL, RR, BB, and TT. An example is illustrated in Figure 4.8.

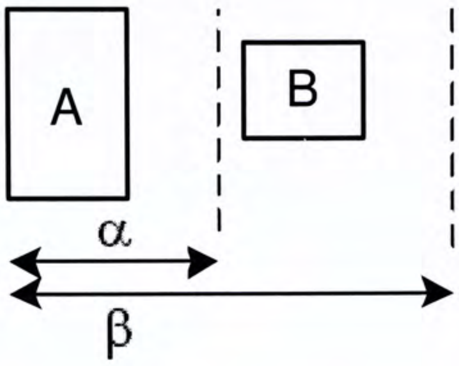


Figure 4.7: Relative Placement Constraint: $h(A,\ B) = [\alpha,\ \beta]$

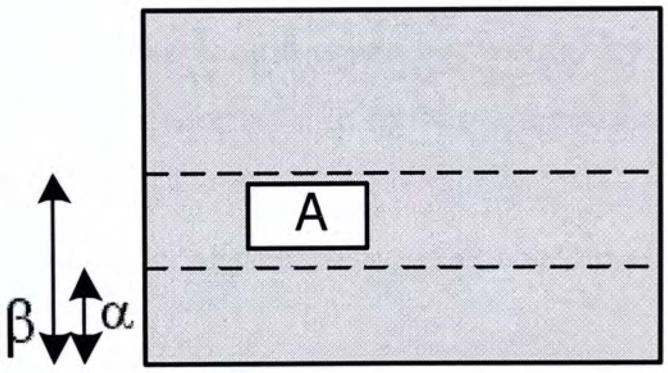


Figure 4.8: Absolute Placement Constraint: $v(BB,\ A) = [\alpha,\ \beta]$

Sequence pair representation is adopted. After modelling different kinds of placement constraints as a collection of relative and absolute placement constraints, they can be enforced by inserting pairs of edges in the constraint graphs. If adding of edges produces positive cycles in the constraint graphs, the packing is infeasible (cannot satisfy all placement constraints). Then, a penalty will be added in the cost function of the simulated annealing process.

Based on the sequence pair representation, the authors of [40] proposed a method to enforce the alignment constraint and some other placement constraints in 2002.

An intuitive idea of deducing the approximate positions of a set of blocks by looking at the sequence pair is proposed in [40]. In the paper, a set of blocks are said to be *H-aligned* if they are abutting with each other horizontally. *V-alignment* can be defined in a similar fashion. After that, the authors defined *strictly ahead* as follow: Given two blocks a and b and a sequence pair (X, Y) $= (X_1aX_2bX_3, Y_1aY_2bY_3)$, a is strictly ahead of b in (X, Y) if and only if the length of the longest common subsequence of $(X_2, Y_2) = 0$.

It is shown that if a set of blocks are *H-aligned*, the relative positions of the blocks in both sequences of the sequence pair should be the same, and the *strictly ahead* relationship should exists between every pair of consecutive neighboring blocks. The method of finding the approximate positions of the blocks by looking at the sequence pair is very helpful. In [1], the authors have made use of this to design an algorithm to solve the bus-driven floorplanning problem.

These kinds of approaches to enforce alignment constraint in a floorplan are again not suitable for solving the bus-driven floorplanning problem, as for a bus to go through a set of blocks, it is not necessary for them to align. Forcing them to align will impose some needless restrictions to the solution. Besides, for a bus to go through a set of blocks, the order in which the blocks are placed is not fixed.

4.2.3 Bus-Driven Floorplanning

In [1], the authors aimed at solving the bus-driven floorplanning problem, based on a simulated annealing framework. Sequence pair representation is used. Each candidate floorplanning solution would be checked in an evaluation step to see if the buses are feasible, i.e., the required set of blocks can be passed through by a 0-bend bus.

The authors has derived necessary conditions for feasible buses. Given a candidate sequence pair, if a bus has to go through a set of blocks B, the relative positions of the blocks in B has to be either the same or reversed in the sequence pair. If more than one buses have to be placed, the orderings between the buses have to be taken into account. The final step of the algorithm is to realize the floorplan, by calculating the coordinates of each blocks and buses. Sometimes the positions of the blocks have to be adjusted in order to let buses to go through.

In 2005, authors in [42] have proposed an algorithm to solve the bus-driven floorplanning problem using the B*-Tree representation. A modified simulated annealing framework is used.

Similar to [1], the authors aim at solving the problem using either horizontal or vertical buses. It is claimed that in a B*-Tree representation, the nodes in a left-skewed sub-tree may satisfy a horizontal bus constraint. Dummy blocks of appropriate heights are then added to guarantee the feasibility of a horizontal bus whose corresponding B*-tree nodes are in a left-skewed sub-tree. Vertical buses can be handled in a similar fashion. After that, the *twisted-bus* structure has to be taken care of (Figure 4.9). Two buses in a twisted-bus structure cannot be placed at the same time. A candidate solution with twisted-bus structures will be discarded.

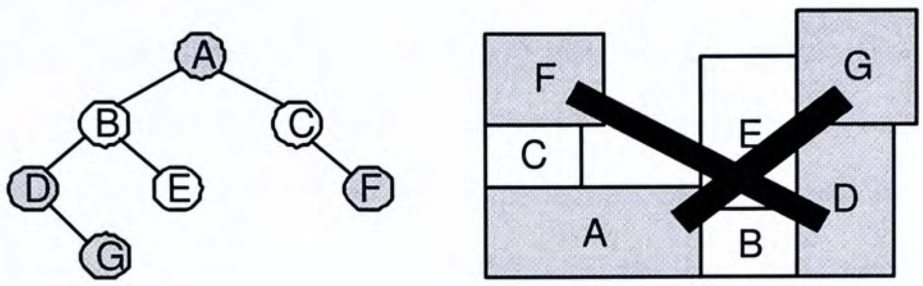


Figure 4.9: A Twisted-Bus Structure.

These paper provided an algorithm to solve the bus-driven floorplanning problem. Nevertheless, one major drawback of their approaches is that, only horizontal and vertical buses are considered and the solution quality will deteriorate if the number of blocks involved in each bus is large, i.e., each bus has to go through many blocks. Our proposed algorithm, which will be discussed in Chapter 5, has made a significant improvement over [1] by allowing 0-bend, 1-bend, and 2-bend buses.

4.3 Summary

In this chapter, some previous work related to the problem bus-driven floorplanning is discussed. The previous work can be divided into three main categories: enforcing abutment constraints, enforcing alignment constraints, and solving the bus-driving floorplanning problem directly.

Many work was done on handling placement constraints in floorplan design. Some of them was proposed to solve the problem of packing rectilinear blocks. In most cases, rectilinear blocks were first partitioned into rectangular sub-blocks. Those sub-blocks were then placed in the floorplan with some placement constraints, like alignment constraints, abutment constraints, etc., in order to get back the original shapes of the rectilinear blocks. However, the abutment constraint alone is not helpful in the bus-driven floorplanning problem, as the blocks involved in a bus are not necessarily abutted. Similarly, alignment constraint is not helpful as it may over-restrict the solution space. Besides, the order in which a bus goes through its blocks is not known beforehand and it is hard to enforce the abutment or alignment constraint.

To solve the bus-driven floorplanning problem, the authors of [1] and [42] have proposed different algorithms, using different floorplan representations. However, both of their works considered only horizontal and vertical buses. The solution quality will deteriorate if the number of blocks involved in a bus is large. Improvements can be made if the buses are allowed to have more bendings.

Chapter 5

Multi-Bend Bus-Driven Floorplanning

The paper [23] of the content of this chapter is included in the proceedings of the *International Symposium of Physical Design (ISPD)* 2005.

5.1 Introduction

Floorplanning is to plan the positions and shapes of a set of modules at the beginning of the design cycle to optimize circuit performance. Interconnect-driven floorplanning is considered to be one of the most important problems in physical design today. As the complexity of chip design increases, the amount of interconnections between different modules on a chip becomes huge. Bus is a collection of wires, which can be used to carry signals among different modules. Bus routing has become more and more important as the complexity of chip design increases. An area-compacted floorplan is not necessarily bus-routable. In order to ease bus routing and avoid unnecessary iterations of the physical design cycle, it would be favourable to incorporate this bus routing problem in the early designing phases.

Bus-driven floorplanning considers the placement of buses. Buses are of different widths and need to go through different sets of modules. Therefore, the positions of the modules will affect the placement of the buses. The objective of the problem is to obtain a bus-routable floorplan such that the area of the chip and the total area of the buses are minimized.

In this chapter, this bus-driven floorplanning problem will be re-visited. Unlike [1], our proposed algorithm allows 0-bend, 1-bend, and 2-bend buses. To have a 1-bend bus, one via is used and thus, it can be considered as a 1-via bus. Experimental results have proven that our algorithm can generate solutions with higher quality especially when the number of blocks in each bus is large. For example, if the buses have to go through more than 10 blocks, [1] is not able to generate any solution while our algorithm can still generate solutions of good quality.

The rest of the chapter is organized as follows. A formal definition of the problem will be given in Section 5.2. After that, an algorithm is proposed to solve the problem, and the details will be discussed in Section 5.3. Experimental results will be presented in Section 5.4. Finally, a summary will be given in Section 5.5.

5.2 Problem Formulation

We assume that buses are routed on two layers, one for horizontal buses and the other for vertical buses. The bus-driven floorplanning problem can be formulated as follows.

Given the following:

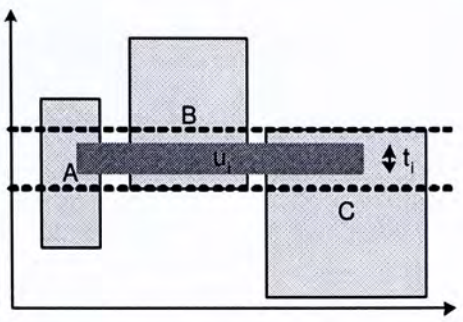


Figure 5.1: Bus u_i Goes Through A, B, and C.

1. A set of n blocks $B = \{b_0, b_1, \cdots, b_{n-1}\}$, where each block b_i is associated with a width w_i and a height h_i, where w_i, $h_i \in \mathbf{R}^+$.

2. A set of m buses $U = \{u_0, u_1, \cdots, u_{m-1}\}$, where each bus u_i has a width t_i, $t_i \in \mathbf{R}^+$, and need to go through a set of blocks B_i, $B_i \subseteq B$.

Our task is to decide the position of each block and the route of each bus, such that all the buses are 0-bend, 1-bend, or 2-bend and each bus u_i goes through all its blocks. There should be no overlapping between any two blocks. As there are only two layers for bus routing, we have to ensure that there is no overlapping between the horizontal (vertical) components of the buses. The goal is to minimize the chip area and the total bus area.

We will define the meaning of "going through" here. For a horizontal component of a bus u_i to go through a set of blocks $\{A, B, C\}$, the vertical overlapping between the blocks has to be greater than or equal to the bus width t_i of u_i. An example is shown in Figure 5.1. The condition for a vertical component of a bus to go through a set of blocks can be defined similarly.

5.3 Methodology

Simulated annealing (SA) will be used to derive a solution. A candidate solution will be evaluated according to (1)the number of buses it can accommodate,

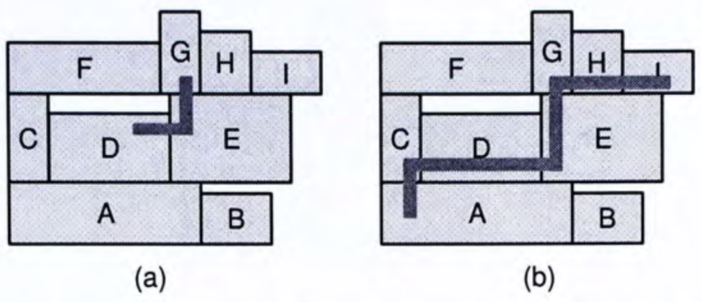


Figure 5.2: (a) A 1-Bend Bus. (b) A 3-Bend Bus.

(2)the total area of the buses, and (3)the total area of the floorplan. There are three main steps to evaluate a solution. The first step is to determine the shapes of the buses by examining the sequence pair. After that, a bus ordering is found such that all feasible buses can be laid out correctly by following this order. Finally, a flooplan is obtained by calculating the coordinates of the blocks and the buses. Details of each step will be presented in the following sections.

5.3.1 Shape Validation

We can deduce the shape of a bus by looking at the sequence pair representation of the floorplan. As we allow buses of at most two bends, buses that cannot be realized in two bends will be considered as infeasible, and will be excluded from further checking. A penalty will be added for each infeasible bus.

An example is shown in Figure 5.2. Consider a sequence pair ($FGHICDEAB$, $ABCDEFGHI$), a bus u_i that need to go through the blocks in $\{D, E, G\}$ can be realized as a 1-bend bus (Figure 5.2a). Another bus u_j that need to go through the blocks in $\{A, C, D, E, G, H, I\}$ will have at least three bends (Figure 5.2b), and it will be marked as infeasible. The aim of this step is to find out all the infeasible buses, and to determine the shape of each feasible bus.

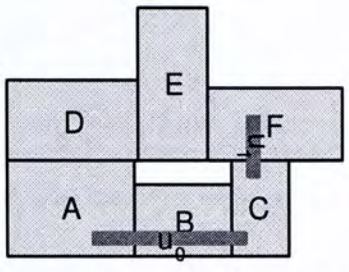


Figure 5.3: Two Valid 0-Bend Buses, $\{A, B, C\}$ and $\{C, F\}$.

Given a bus u_i that need to go through $B_i = \{b_1, b_2, \cdots, b_k\}$, we will first extract those blocks in B_i from the sequence pair, without altering their relative positions. For example, if we are checking a bus that goes through the blocks in $\{A, B, E\}$ from the sequence pair $(ADBCE,\ EBCAD)$, we will first extract $sp_i = (ABE,\ EBA)$ from the sequence pair, where sp_i denotes the extracted sequence pair for bus u_i. Then, we will work on sp_i to check whether u_i can be realized as a 0-bend, 1-bend, or 2-bend bus one after another.

0-Bend Bus Checking

A 0-bend bus is actually a horizontal bus or a vertical bus. For a bus u_i to be 0-bend, the orders of the blocks in the two sequences of sp_i have to be either the same (horizontal bus) or reversed (vertical bus). Let (α, β) be the sequence pair of sp_i, α and β are in reversed order if $\alpha = \beta^R$, where X^R is the reverse of string X. For example, given a sequence pair $(DEFABC,\ ABCDEF)$ and a bus u_0 that has to go through the blocks in $\{A, B, C\}$, the first step is to extract the corresponding blocks from the sequence pair: $sp_0 =(ABC,\ ABC)$. As the blocks appear in the same order in both sequences, it can be concluded that u_0 can be realized as a 0-bend horizontal bus. For another bus u_1 that has to go through the blocks in $\{C, F\}$, the extracted sp_1 is $(FC,\ CF)$. As the blocks appear in reversed order in the two sequences, it can be realized as a 0-bend vertical bus. This example is illustrated in Figure 5.3.

1-Bend Bus Checking

1-bend bus is also called L-shaped bus. For a bus to be 1-bend, a necessary condition is that it consists of one vertical component and one horizontal component. This can be checked easily by identifying the longest common subsequence (LCS) in sp_i first, and then check if the remaining blocks (after removing the blocks in the LCS) in the two sequences are in reversed order.

We have to identify the longest common subsequence to form the horizontal component of an L-shaped bus. It must be the LCS but not any common subsequence in sp_i because we have proved that if taking the longest common subsequence as the horizontal component fails to form a valid L-shape, taking any other shorter subsequences will also fail. Let l_1 be the longest common subsequence of sp_i and l_2 be another common subsequence of shorter length. We can analyze the situation by looking at two different cases. The first case is that l_2 is not a substring of l_1. Then, a valid L-shape can never be formed with l_2 as the horizontal component because there exist at least two blocks n_1 and n_2 which are in l_1 but not in l_2, and these two blocks must be in a left-right relationship with each other. This implies two separate horizontal components and thus, a valid L-shape cannot be formed. Another case is that l_2 is a substring of l_1. Similarly, choosing l_2 as the horizontal component will prevent a valid L-shape to be formed as those blocks in l_1 must be in left-right relationship with each other. Therefore, we will pick the longest common subsequence as the horizontal component.

If there exist more than one longest common subsequences l_1 and l_3, picking either one of them will be the same. Let's consider three different cases according to the number of blocks in l_1 but not in l_3. The first case is that there exist more than one blocks in l_1 but not in l_3 (i.e., there exist more than one blocks in l_3 but not in l_1). Then, the blocks in l_1 but not in l_3 will form a horizontal component, and so as the blocks in l_3 but not in l_1. Thus, a valid L-shape cannot be formed no matter which one we pick. The second case is that there is only one block x that is in l_1 but not in l_3 (i.e., there is another block y that is in l_3 but not in l_1), and that block appears in the middle of l_1, i.e., x is neither the first nor the last block in l_1. Note that the position of x in l_1 must be the same as that of y in l_3. In this case, a T-shape (not L-shape) will be formed if we take l_1 as the horizontal component, as y will be in an upper-lower relationship with x. Notice that we cannot take l_3 as the horizontal component neither in this second case for a similar reason. The last case is that there is only one block x that is in l_1 but not in l_3, and x is the first block or the last block of l_1. A valid L-shape may be formed as x can participate in the vertical component and act as a 'joint' of the two components. In the last case, picking either l_1 and l_3 will be the same. In the following steps, we will regard the first and the last block of the longest common subsequence as in the vertical component and will keep them for checking whether the vertical component is on the left or on the right of the horizontal component.

Note that even if a bus is consisted of one vertical component and one horizontal component only, there are still several possibilities. The blocks may be in T-shape or +-shape which we consider as invalid. Let $\{a_0, a_1, \cdots, a_x\}$ be the set of blocks that form the vertical component, and $\{b_0, b_1, \cdots, b_y\}$ be the set of blocks that form the horizontal component. If there exists a block b_i that has to be on the left of a_j for some $j \in \{0, 1, \cdots, x\}$, and a block b_k that has to be on the right of a_l for some $l \in \{0, 1, \cdots, x\}$, this bus is in T-shape (or $\perp$-shape or +-shape) and is invalid. Similarly, if there exists a block a_i that has to be on top of b_j for some $j \in \{0, 1, \cdots, y\}$, and a block a_k that has to be below b_l for some $l \in \{0, 1, \cdots, y\}$, this bus is in $\vdash$-shape (or $\dashv$-shape or +-shape) and is also invalid.

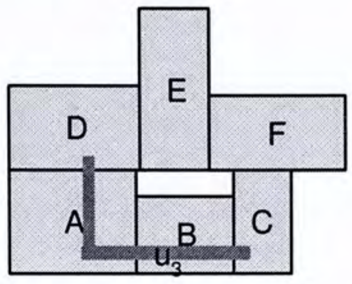


Figure 5.4: A Valid 1-Bend Bus $\{A, B, C, D\}$

Let's look at an example. Given a sequence pair $(DEFABC,\ ABCDEF)$ and a bus u_3 that has to go through the blocks $\{A, B, C, D\}$, the first step is to extract the corresponding blocks $sp_3 = (DABC,\ ABCD)$ from the sequence pair. As it failed the 0-bend checking, the next step is to check if it can be realized as a 1-bend bus. The LCS of sp_3 is ABC, so ABC will be taken as the horizontal component of u_3 and B will be removed from sp_3. Then we have to check whether the remaining block D can form a vertical component with the block A or C. As the blocks A and D appear in reversed order in sp_3, AD can form the vertical component of u_3 (Note that C and D also appear in reversed order in sp_3 and we can pick either AC or AD). After checking, u_3 is classified as a valid 1-bend bus. This example is illustrated in Figure 5.4.

Let's look at another example. given the same sequence pair $(DEFABC,$ $ABCDEF)$ and another bus u_4 that has to go through the blocks in $\{A, B, E, F\}$, we first extract the corresponding blocks $sp_4 = (EFAB,\ ABEF)$ from the sequence pair. The LCS is AB or EF. As there exist more than one longest common subsequence and there are more than one different symbols between them, it is not a valid 1-bend bus and will proceed to the 2-bend checking. This example is illustrated in Figure 5.5.

In this 1-bend checking, some buses may be identified as T-shaped but we will not mark it as infeasible yet since it may form a valid 2-bend bus by

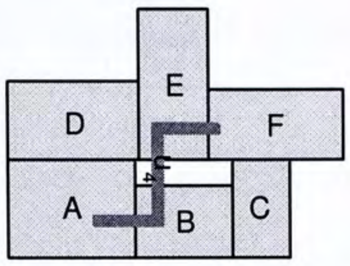


Figure 5.5: Bus u_4 Cannot Be Realized as A 1-Bend Bus.

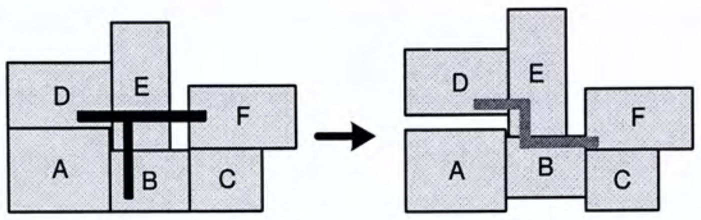


Figure 5.6: In Some Cases, A T-Shaped Bus Can Be Changed into A Valid 2-Bend Bus.

adjusting the positions of some blocks. An example is illustrated in Figure 5.6.

2-Bend Bus Checking

If the bus is found to be neither 0-bend nor 1-bend, we will check whether it is a 2-bend bus. There are several kinds of 2-bend buses, Z-shape, mirrored Z-shape, C-shape, or mirrored C-shape. There will be two horizontal (vertical) components and one vertical (horizontal) component in the bus, denoted by HVH or VHV respectively. Assuming the case of HVH, we will first identify the vertical component of the bus. Let the extracted sequence pair sp_i of bus u_i be (α, β), where α and β are strings of blocks. The vertical component can be found by finding the longest common subsequence in (α, β^R), where β^R denotes the reverse of the string β.

Similar to 1-bend checking, the first block and the last block of the longest common subsequence will be kept for horizontal component checking. Besides, we have to pick a longest common subsequence but not any other shorter subsequence, and if there are more than one longest common subsequences, picking any one of them will do. The argument is similar to that in 1-bend checking.

After identifying the vertical component, we will classify the remaining blocks of the bus into different relationships with the vertical component. For example, block A from the bus u_i with extracted sequence pair $sp_i = (ABCDEF, FEDABC)$ will be classified as in the set *Upper*, as A is on top of all the blocks in the vertical component. On the other hand, block F will be classified as in the set *Lower*, as F is below all the blocks in the vertical component. We can deduce these relationships easily from the sequence pair. There are totally eight types of *position sets*: (1) *Upper*, (2) *UpperLeft*, (3) *Left*, (4) *LowerLeft*, (5) *Lower*, (6) *LowerRight*, (7) *Right*, and (8) *UpperRight*.

There are four valid shapes for the case of HVH: Z-shape, mirrored Z-shape, C-shape, and mirrored C-shape. In order to form a valid shape, some of the *position sets* have to be empty. For example, to form a mirrored Z-shape, there should be no block in the upper-left and lower-right directions of the vertical component. Thus, the sets *UpperLeft* and *LowerRight* have to be empty. The blocks in the set *Upper*, *UpperRight*, and *Right* will form one horizontal component, and the blocks in the set *Lower*, *LowerLeft*, and *Left* will form another horizontal component. Details are shown in Figure 5.7. The last step is to check both horizontal components to ensure that the blocks in each component can indeed align horizontally, i.e., the blocks appear in the same order in both sequences of sp_i.

The shape validation step for 0-bend, 1-bend, and 2-bend buses can be incorporated into one whole process. The overall algorithm is shown in Figure 5.8.

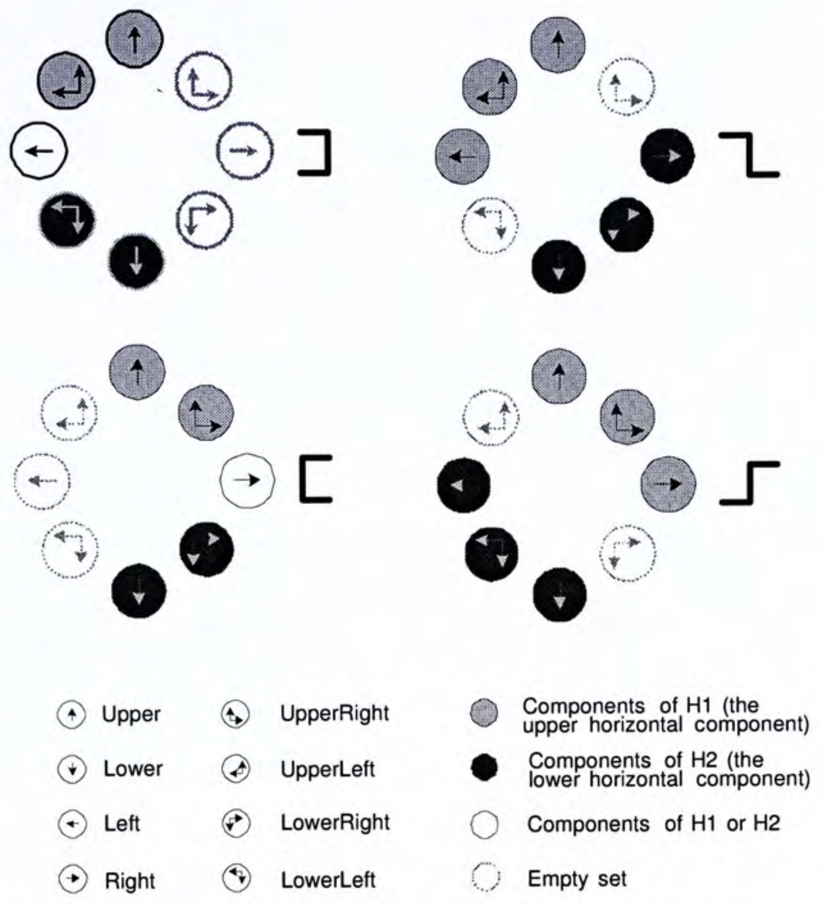

Upper
Lower
Left
Right
UpperRight
UpperLeft
LowerRight
LowerLeft
Components of H1 (the upper horizontal component)
Components of H2 (the lower horizontal component)
Components of H1 or H2
Empty set

Figure 5.7: The Necessary Conditions for The Position Sets to Form A Valid 2-Bend Shape.

5.3.2 Bus Ordering

In this step, we aim at determining an ordering between the valid buses, and removing those that have conflicts with some other buses. For example, given a sequence pair ($CADB$, $ACBD$), block C has to be placed above block A according to the order in the sequence pair, so any horizontal bus going through block C has to be placed above any horizontal bus going through block A. This kind of constraint is called bus ordering constraint.

However, some ordering constraints may be contradictory to each other. An example is shown in Figure 5.11. In this example, block A is on the left of block B according to the sequence pair, so any vertical bus going through A has to be placed on the left of any vertical bus going through block B. Similarly, block C is on the left of block D and thus, any vertical bus going

SHAPE_VALIDATION (int i)

```
$k \leftarrow$ number of blocks that bus $u_i$ has to go through
Extract $sp_i$ from the sequence pair
Find the longest common subsequence $lcs_i$ of $sp_i$
IF $|lcs_i|$ = 1 OR $|lcs_i|$ = $k$
    Mark as 0-bend
    $result \leftarrow$ SUCCESS
ELSE
    Put the remaining blocks into position sets
    $result \leftarrow$ ONE_BEND_CHECK($i$)
    IF $result$ = FAIL
        $result \leftarrow$ TWO_BEND_CHECK_VHV($i$)
        IF $result$ = FAIL
            Reverse the first sequence in $sp_i$
            Find the longest common subsequence of $sp_i$
            Put the remaining blocks into position sets
            $result \leftarrow$ TWO_BEND_CHECK_HVH($i$)
        END IF
    END IF
END IF
RETURN $result$
```

Figure 5.8: Pseudo Code of Shape Validation.

ONE_BEND_CHECK (int i)

```
$result \leftarrow$ FAIL
IF |Right| = 1 (The vertical component must be on the left)
    IF $|UpperRight|=0 \wedge |LowerRight|=0 \wedge |Lower|=0 \wedge |LowerLeft|=0$
        IF Upper $\cup$ UpperLeft can form a vertical component
            Mark as ⌞-shape and $result \leftarrow$ SUCCESS
        END IF
    ELSE IF $|UpperLeft|=0 \wedge |Upper|=0 \wedge |UpperRight|=0 \wedge |LowerRight|=0$
        IF Lower $\cup$ LowerLeft can form a vertical component
            Mark as ⌜-shape and $result \leftarrow$ SUCCESS
        END IF
    END IF
ELSE IF |Left| = 1 (The vertical component must be on the right)
    IF $|UpperLeft|=0 \wedge |LowerLeft|=0 \wedge |Lower|=0 \wedge |LowerRight|=0$
        IF Upper $\cup$ UpperRight can form a vertical component
            Mark as ⌟-shape and $result \leftarrow$ SUCCESS
        END IF
    ELSE IF $|UpperRight|=0 \wedge |Upper|=0 \wedge |UpperLeft|=0 \wedge |LowerLeft|=0$
        IF Lower $\cup$ LowerRight can form a vertical component
            Mark as ⌝-shape and $result \leftarrow$ SUCCESS
        END IF
    END IF
END IF
RETURN $result$
```

Figure 5.9: Pseudo Code of 1-Bend Checking.

```
TWO_BEND_CHECK_VHV (int i)
    result ← FAIL
    IF |UpperRight| = 0 AND |LowerLeft| = 0
        IF the blocks in Upper, UpperLeft, Left can be vertical AND
        the blocks in Lower, LowerRight, Right can be vertical
            Mark as 2-bend and result ← SUCCESS
        END IF
    ELSE IF |UpperLeft| = 0 AND |LowerRight| = 0
        IF the blocks in Upper, UpperRight, Right can be vertical AND
        the blocks in Lower, LowerLeft, Left can be vertical
            Mark it as 2-bend and result ← SUCCESS
        END IF
    ELSE IF |LowerLeft| = 0 AND |LowerRight| = 0 AND |Lower| = 0
        IF the blocks in Upper, UpperLeft, Left can be vertical AND
        the blocks in Upper, UpperRight, Right can be vertical
            Mark it as 2-bend and result ← SUCCESS
        END IF
    ELSE IF |UpperLeft| = 0 AND |UpperRight| = 0 AND |Upper| = 0
        IF the blocks in Lower, LowerLeft, Left can be vertical AND
        the blocks in Lower, LowerRight, Right can be vertical
            Mark it as 2-bend and result ← SUCCESS
        END IF
    END IF
    RETURN result
```

Figure 5.10: Pseudo Code of 2-Bend Checking.

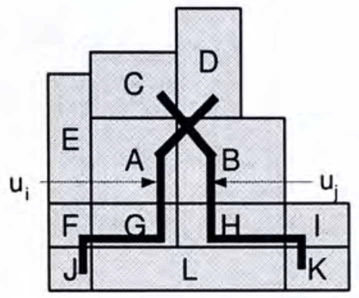


Figure 5.11: Bus u_i Has to Be Placed on The Left of u_j and Bus u_j Has to Be Placed on The Left of Bus u_i.

through C has to be placed on the left of any vertical bus going through D. Problem will occur if there are two 2-bend buses u_i and u_j, where a vertical component of u_i has to go through block A and D, and a vertical component of u_j has to go through block B and C. These two vertical components have to be placed on the left hand side of each other, which is impossible. This step aims at removing the least number of buses such that the remaining buses do not have any conflict with each other. For simplicity, our discussion is limited to the horizontal components of the buses, where the case for the vertical components can be derived similarly.

Assuming that buses are routed on two layers, one layer for horizontal buses and the other for vertical buses. We can consider the constraints between horizontal components and the constraints between vertical components separately. For 1-bend or 2-bend buses, we will first break them down into two or three 0-bend components respectively before checking the ordering constraints (Figure 5.12).

For horizontal buses, we use a graph $G = (V, E)$ to determine whether all the ordering constraints can be satisfied. Each vertex in V represents a 0-bend component, and $E = \{(v_i, v_j)|$ component v_i has to be placed above component $v_j.\}$. In order to check if (v_a, v_b)$\in E$, we will first extract sp_{ab} from the sequence pair, where sp_{ab} contains only the blocks in u_a and u_b. For

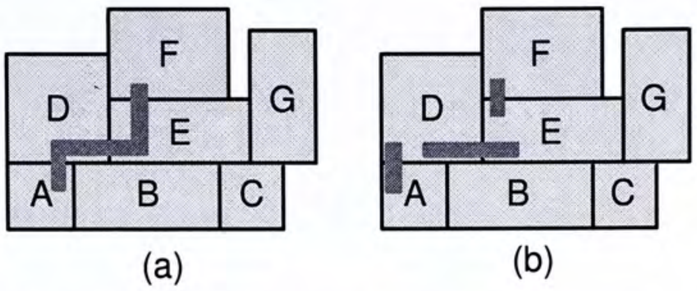


Figure 5.12: A 2-Bend Bus is Broken Down into Three 0-Bend Components for Checking The Ordering Constraints.

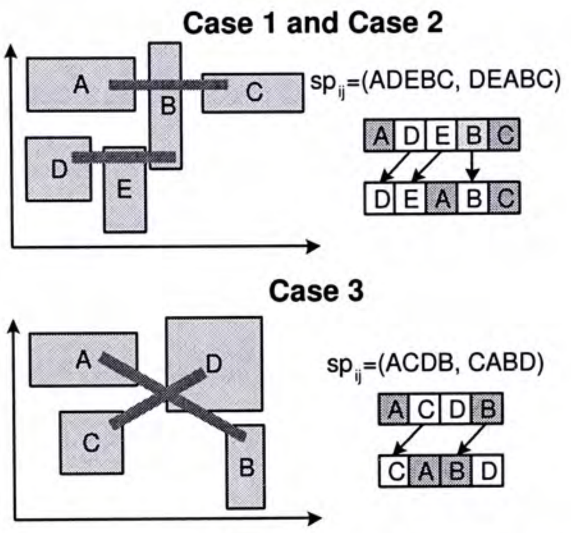


Figure 5.13: Different Cases of The Bus Ordering Constraint.

example, if the sequence pair is $(ABCDEF,\ DEACBF)$, and u_a has to go through block A and B and u_b has to go through block C and D, the extracted sp_{ab} will be $(ABCD,\ DACB)$.

Let m be a block, $s_1[m]$ denotes the position of block m in the first sequence of sp_{ab}, e.g., $s_1[A]$ in the above example is one. Similarly, $s_2[m]$ is the position of block m in the second sequence of sp_{ab}. In the above example, $s_2[A]$ is two. Let $B_a(B_b)$ be the set of blocks that $u_a(u_b)$ has to go through.

After computing the $s_1[m]$ and $s_2[m]$ for each related block m, we will check if sp_{ab} falls into one of the following three cases (Figure 5.13):

1. If $\forall x \in B_a$, $s_1[x] \geq s_2[x]$, and $\exists y \in B_a$, $s_1[y] > s_2[y]$, then u_a is below

u_b. Thus, $(v_a, v_b) \in E$.

2. If $\forall x \in B_b$, $s_1[x] \geq s_2[x]$, and $\exists y \in B_b$, $s_1[y] > s_2[y]$, then u_b is below u_a. Thus, $(v_b, v_a) \in E$.

3. If $\exists x \in B_a$, $s_1[x] > s_2[x]$, and $\exists y \in B_b$, $s_1[y] > s_2[y]$, then contradiction occurs, as u_a cannot be above u_b and below u_b at the same time. Thus, $(v_b, v_a) \in E$ and $(v_a, v_b) \in E$.

As some of the buses cannot be placed at the same time, our aim in this step is to remove the least number of buses such that all the remaining buses can be placed. Besides, we aim at finding an ordering for the remaining buses such that they can be placed one after another successfully in a bottom-up (left-right) fashion according to the order. To do so, we have to examine the graph G_h. Contradiction exists if cycle presences. So the first step is to check whether cycles exist in G_h. If there are cycles, we want to remove the least number of nodes (buses) to make the graph acyclic. However, this Node-Deleting Problem is proven to be NP-complete [1]. Our heuristic to solve the problem is to keep on removing the node with the highest degree (in-degree plus out-degree), until the graph is acyclic.

Assume that a 2-bend bus u_i is broken into three 0-bend components u_1, u_2, and u_3, where u_1 and u_3 are horizontal and u_2 is vertical. When processing the horizontal buses, a graph G_h is built. If u_1 is selected to be removed in order to make G_h acyclic, u_3 in the horizontal graph and u_2 in the vertical graph have to be removed as well. This is obvious since we should not keep partial bus components in the solution, if some components of the bus are already marked as invalid.

In some cases, bending can help to resolve conflicts in the ordering constraint graph G_v and G_h. An example is shown in Figure 5.14. In the example

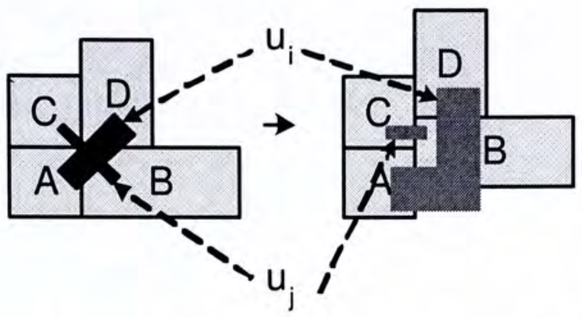


Figure 5.14: Adding Bend to Resolve Bus Ordering Conflict.

u_i and u_j are horizontal buses that contradict with each other. Changing u_i from 0-bend to 1-bend can resolve the conflict without removing any bus from the graph. However, this technique of adding bends to a bus to resolve conflict can only be used for buses that are 0-bend or 1-bend originally, so that one more bend can be added to resolve the conflict by the method as illustrated in Figure 5.14. After obtaining an acyclic graph, an ordering of the remaining buses can be obtained from a topological sort of G_h.

5.3.3 Floorplan Realization

The final step to evaluate a candidate solution is to realize the floorplan, i.e., obtaining the coordinates of the blocks and buses, to determine the chip area and the total bus area. After the previous checkings, all the invalid buses are removed, and a correct bus ordering is found. Based on those information, we can compute the coordinates of all the blocks and valid buses, and thus the chip area and total bus area. In order to obtain the coordinates of the blocks, we used the algorithm FAST-SP in [33] to construct a floorplan from the sequence pair.

We use the same approach as in [1], which can be described in brief as follows. The following process repeats $O(m)$ times, where m is the total number of valid buses. Note that all 1-bend and 2-bend buses will have been broken

```
BASIC_ALIGNMENT_H (int $i$)
1    $y_{max} \leftarrow max\{y_k : u_i$ goes through block $k\}$
2    FOR all blocks $j$ $u_i$ goes through
3        IF $y_{max} + t_i - h_j > y_j$
4            $y_j \leftarrow y_{max} + t_i - h_j$
5        END IF
6    END FOR
```

Figure 5.15: Pseudo Code of The Basic Alignment Step for Horizontal Buses.

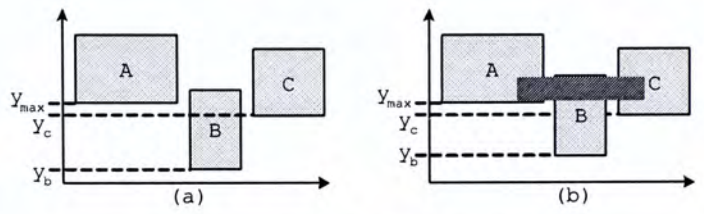


Figure 5.16: (a) y_{max}, y_b, and y_c are Calculated Correspondingly. (b) y_b Has to Be Moved Up to Let The Bus Go Through.

down into 0-bend buses for processing. Let's consider horizontal buses only. In iteration i, bus u_i will be processed. The coordinates of the blocks that u_i goes through will be computed first. Then, the position of u_i will be calculated by performing some basic alignment steps between the blocks that u_i goes through. These basic alignment steps for horizontal buses are shown in Figure 5.15. An example is shown in Figure 5.16.

After doing the basic alignment steps, we will check if u_i overlaps with any previously placed bus. If so, u_i will be moved up and the coordinate y_{u_i} will be updated. If u_i is moved up, all the blocks that u_i goes through must be deleted again. We may need to move some of them up in some cases.

5.3.4 Simulated Annealing

Simulated Annealing (SA) is used to search for a good solution. In this section, the set of moves and the cost function used in the SA will be discussed.

Moves

To change from one candidate solution to another, we use two operations, swap and rotate.

1. **Swap** is to exchange the positions of two blocks in either the first sequence or the second sequence. This can be done in constant time.
2. **Rotate** is to exchange a block height with its width. This can be done in constant time.

Cost Function

As mentioned before, the aim of the problem is to (1)accommodate all the buses, (2)minimize the total area of the buses, and (3)minimize the area of the floorplan. Bus area is included in the cost function as bus is actually a collection of wires, and it will be favorable to have the total bus area (interconnect resources) as small as possible. Thus, the cost function is defined as follows.

$$Cost = \alpha \cdot A + \beta \cdot B + \gamma \cdot I$$

where A is the chip area, B is the total bus area, I is the number of invalid bus, and α, β, and γ are parameters that can be specified by the users.

In this bus-driven floorplanning problem, we focused on fitting all the buses in a compact floorplan solution. Other aspects like the total wire length and routing congestion can also be considered by including more terms in the cost function.

5.3.5 Soft Block Adjustment

In order to compare with the results presented in [1], we have added the feature of 'soft block adjustment'. The adjustment is the same as that in [1]. This step makes use of the fact that the width and height of a block can be altered as long as the area is unchanged and the dimension is constrained by an aspect ratio bound. The process is again done by simulated annealing. The cost function is the same as before. In each pass, a block lying on a critical path will be selected, and the width or height of it will be changed a little bit. Then, the floorplan realization step is repeated to obtain a new chip area and total bus area. Note that if an originally valid bus is made invalid, the candidate solution will be discarded. Besides, when changing a block width or height, the aspect ratio constraint has to be obeyed.

5.4 Experimental Results

The proposed algorithm was implemented using the C++ language and the experiments were conducted using an Intel Xeon (2.2 GHz) machine with 1G memory. The test cases are derived from the MCNC benchmarks for floorplanning. In order to compare with the results presented in [1], the same test cases are tried using our proposed algorithm and all the experiments (including those of [1]) are run on the same machine. The ratio of α:β:γ is set to be 1:1:1. The results are listed in Table 6.4. Comparing with the results of [1], the dead space of the floorplan obtained by our algorithm can be reduced on average.

To demonstrate the importance of having 1-bend and 2-bend buses, we have created another set of test cases based on the ami33 and ami49 benchmarks. In these test cases, each bus will go through at least ten blocks. The

File	No. of Blocks	No. of Buses	Average/Max. No. of Blocks in a Bus
apte	9	5	2.60 / 3
xerox	10	6	2.50 / 3
hp	11	14	2.29 / 3
ami33-1	33	8	4.17 / 6
ami33-2	33	18	2.39 / 4
ami49-1	49	9	4.00 / 6
ami49-2	49	12	3.58 / 6
ami49-3	49	15	3.53 / 6

Table 5.1: Data Set One.

File	No. of Blocks	No. of Buses	Average/Max. No. of Blocks in a Bus
ami33-3	33	1	10.00 / 10
ami33-4	33	3	10.00 / 10
ami33-5	33	5	10.00 / 10
ami49-4	49	1	15.00 / 15
ami49-5	49	3	11.67 / 15
ami49-6	49	4	11.25 / 15

Table 5.2: Data Set Two.

	[1]		Our Work		Comparison*	
	Time (s)	Dead Space	Time (s)	Dead Space	Time	Dead Space
apte	15	0.72%	30	0.48%	+100%	-33.33%
xerox	15	0.95%	35	0.42%	+133.33%	-55.79%
hp	33	0.62%	51	0.29%	+54.55%	-53.23%
ami33-1	11	0.94%	93	1.00%	+745.45%	+6.38%
ami33-2	92	1.27%	144	1.19%	+56.62%	-6.30%
ami49-1	16	0.85%	71	0.56%	+343.75%	-34.12%
ami49-2	302	0.84%	713	0.58%	+136.09%	-30.95%
ami49-3	285	1.09%	865	0.60%	+203.51%	-44.95%
				Average:	+221.65%	-31.54%

*It is calculated by $((y_1 - y_0)/y_0) * 100\%$, where y_0 and y_1 are the time (dead space) obtained by [1] and by our algorithm respectively.

Table 5.3: Results of Data Set One.

	[1]		Our Work		Comparison	
	Time (s)	Dead Space	Time (s)	Dead Space	Time	Dead Space
ami33-3	86	1.81%	32	1.01%	-62.79%	-44.20%
ami33-4	>10^3	–	92	1.90%	–	–
ami33-5	>10^3	–	95	3.80%	–	–
ami49-4	73	19.34%	88	0.63%	+20.55%	-96.74%
ami49-5	>10^3	–	261	1.17%	–	–
ami49-6	>10^3	–	140	2.19%	–	–
		Average:	118	1.78%		

Table 5.4: Results of Data Set Two.

results are shown in Table 6.5. For this data set, the approach in [1] is not able to generate any solution for most of the test cases, while our algorithm can still generate solution with high quality (with average dead space of 1.8% only). We can see that our algorithm can perform much better. As their approach allows only 0-bend bus, it is very difficult to accommodate several buses that go through many blocks.

5.5 Summary

In this chapter, an algorithm to solve the bus-driven floorplanning problem allowing 0-bend, 1-bend, and 2-bend buses is proposed. Experimental results show that our approach is effective. The presence of 1-bend and 2-bend buses is important especially when the number of blocks that a bus goes through is large. It is difficult to find a solution if only 0-bend bus is allowed in those cases.

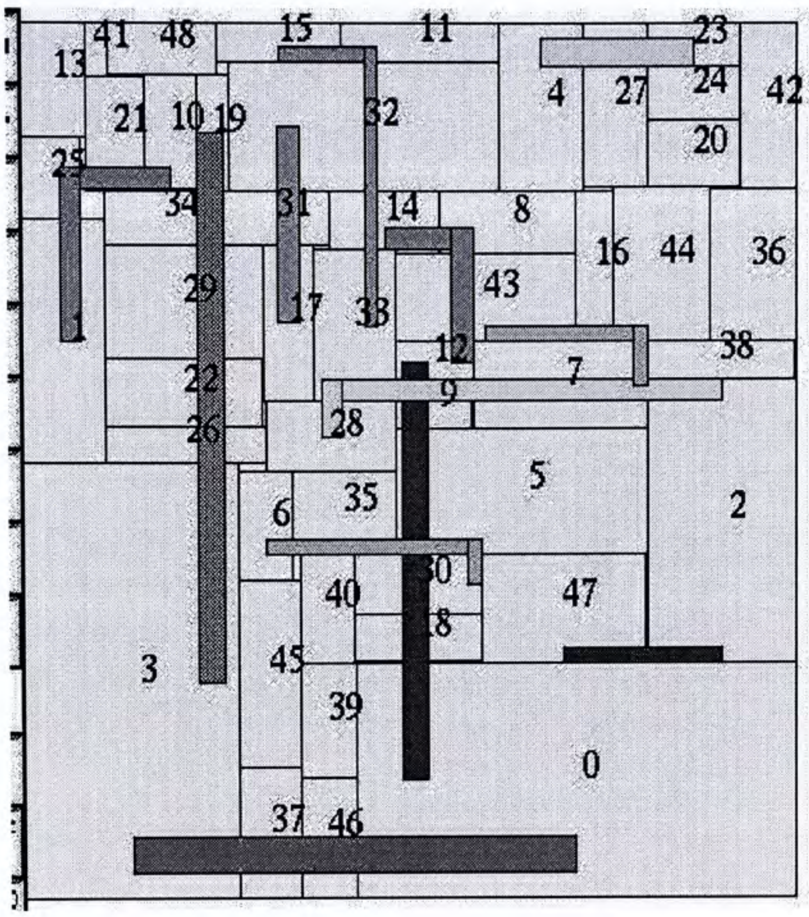


Figure 5.17: Result Packing of ami49-2.

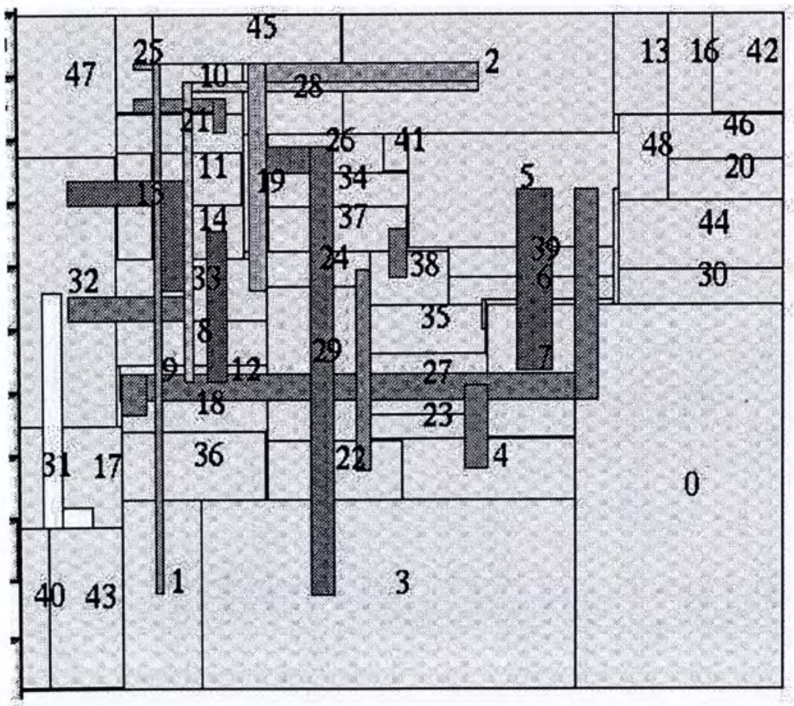


Figure 5.18: Result Packing of ami49-3.

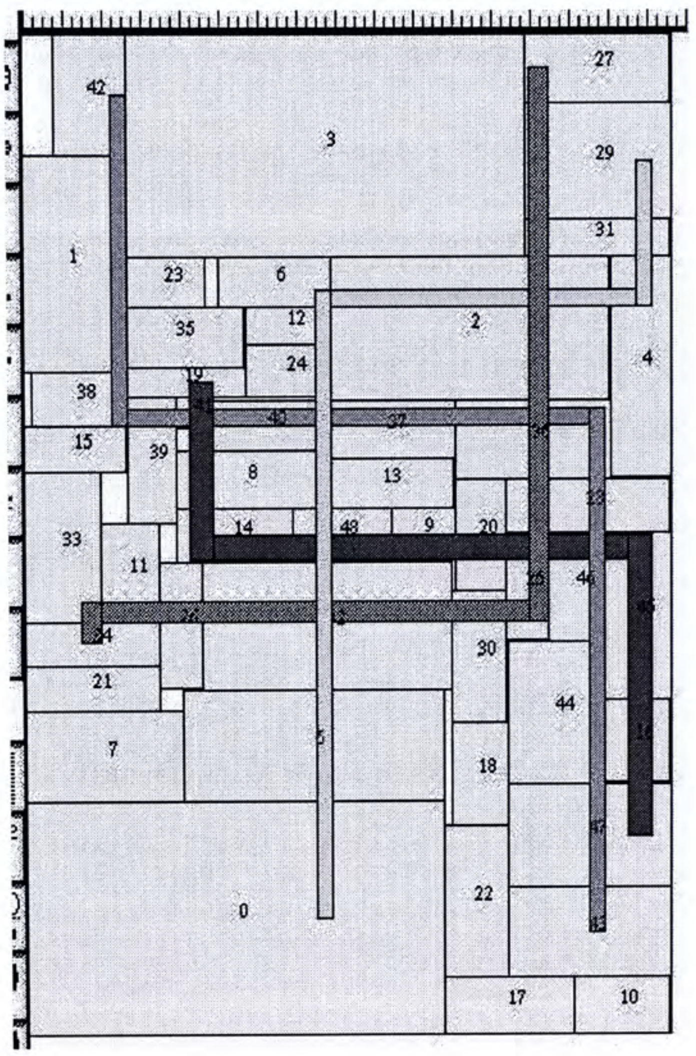


Figure 5.19: Result Packing of ami49-6.

Chapter 6

Bus-Driven Floorplanning for 3D Chips

The paper on the content of this chapter is submitted to the 11th *Asia and South Pacific Design Automation Conference (ASP-DAC)* 2006.

6.1 Introduction

In modern IC designs, the growing number of long on-chip wires is a byproduct of the increasing circuit complexity. As circuits are expected to perform more complicated functions, the number of interconnects involved has increased inevitably. Interconnect delay has dominated over gate delay as technology advances into the deep submicron era. Timing constraints have become more and more difficult to be met with this huge number of interconnects involved. Interconnect-driven floorplanning becomes one of the top ten physical design problems [43]. 3D chip is a solution to these problems. It can greatly reduce interconnect lengths.

A 3D chip is not a "true" 3D structure where each block is associated with three dimensions. It is actually a chip with more than one silicon layers to place modules. A 3D floorplan is also known as a multi-layer floorplan. Therefore, those traditional 3D representations cannot be used directly here to solve the multi-layer floorplanning problem. There is not much work done on this 3D floorplan representation problem and we would like to propose an elegant 3D floorplan representation for multi-layer circuit design. Moreover, we have studied the bus-driven floorplanning problem in 3D chips. We will propose a method to align blocks on different layers of a 3D floorplan, by adding edges into the 3D floorplan representation.

In this chapter, the floorplanning problem in 3D circuit design will be discussed. This chapter is organized as follows: In Section 6.2, the problem will be defined formally. After that, a floorplan representation proposed for 3D circuit design will be discussed in Section 6.3, followed by our proposed method to align blocks on different layers of a 3D floorplan. Then some experimental results will be presented in details in Section 6.6. A conclusion will be drawn in Section 6.7.

6.2 Problem Formulation

A formal definition of the 3D floorplanning problem with bus aligment is given as follows:

Problem: 3D Floorplanning with Bus Alignment Given a set of n modules $\{M_1, M_2, \cdots, M_n\}$ and a value K that represents the number of layers and a set of m buses $U = \{u_0, u_1, \cdots, u_{m-1}\}$. Each module M_i is associated with an area A_i and two aspect ratio bounds r_i and s_i such that $r_i \leq h_i/w_i \leq s_i$, where h_i and w_i are the height and the width of module i respectively. Each bus i is required to go through a set of blocks B_i, $B_i \subseteq M$. We want to find a feasible 3D floorplan F, i.e., the coordinates (x_i, y_i) and the dimensions (h_i, w_i) of modules i, and the layer l_i on which module i lies, where $1 \leq l_i \leq K$, such that if the blocks in B_i where $1 \leq i \leq m$ are placed in k different layers $l_{i,\pi(1)}, l_{i,\pi(2)}, \cdots, l_{i,\pi(k)}$ where $2 \leq k \leq K$ $(l_{i,\pi(1)} < l_{i,\pi(2)} < l_{i,\pi(k)})$, there is at least one block on layer $l_{i,\pi(j)}$ aligning with a block on layer $l_{i,\pi(j+1)}$ in the z-direction for $1 \leq j \leq K-1$. There should be no overlapping between modules on each layer, and the circuit performance should be optimized.

6.3 The Representation

6.3.1 Overview

In this section, a multi-layer floorplan representation *Layered Transitive Closure Graph (LTCG)* is proposed. It is based on a 2D representation called Transitive Closure Graph (TCG) [32]. TCG describes the geometric relationships between different modules according to two constraint graphs C_H and C_V. Apart from traditional 2D floorplans, multi-layer floorplans involve layers as well. In LTCG, there are also two constraint graphs but each block is assigned to one layer. Blocks on the same layer cannot overlap with each other. To achieve this, blocks on the same layer must have a horizontal or vertical relationship with each other. However, blocks on different layers do not have this constraint. Therefore, for two blocks on different layers, they may overlap in the x or y directions.

Simulated annealing is used to search for a good solution. A set of moves are designed to change one candidate solution to another. LTCG is capable for handling block alignment effectively in 3D floorplans. We can align blocks on different layers by adding edges into the LTCG. Details of LTCG and our 3D floorplanner will be presented in the following sections.

6.3.2 Review of TCG

Transitive Closure Graph (TCG) was first proposed in 2001 in [32]. There are two graphs $C_h = (V,\ E_h)$ and $C_v = (V,\ E_v)$ in TCG to represent a 2D floorplan where V is a set of vertices representing the blocks, namely horizontal transitive closure graph and vertical transitive closure graph respectively. For two blocks x and y, if x is on the left of y, a directed edge $(x,\ y)$ is added to C_h. If x is below y, a directed edge $(x,\ y)$ is added to C_v.

TCG have the following three properties [32]:

1. C_h and C_v are acyclic.
2. Each pair of nodes must be connected by exactly one edge in C_h or C_v.
3. The transitive closures of C_h and C_v are equal to themselves respectively, where the transitive closure of a graph $G = (V,\ E)$ is defined as a graph $G' = (V,\ E')$ where $E' = \{(n_i,\ n_j)$: there is a path from node n_i to node n_j in $G\}$

An edge $(x,\ y)$ is said to be a *reduction edge* if there exists no other path from block x to block y in the same graph. Please note that if we want to reverse an edge direction or move an edge from a graph to another during a perturbation, the selected edge must be a reduction edge. Otherwise, cycle may be resulted.

Realization of a floorplan from its TCG representation can be done in $O(n^2)$ time, by performing a longest path search for each vertex in both graphs. The size of the solution space is $O((n!)^2)$.

6.3.3 Layered Transitive Closure Graph (LTCG)

Based on TCG, we proposed a multi-layer floorplan representation namely Layered Transitive Closure Graph (LTCG). The representation consists of two main components: the layer information and the transitive closure graphs. The layer information stores the layer assignment of each block. The two transitive closure graphs show the topological relationship between the blocks.

We denote the two transitive closure graphs in LTCG by $G_h = (V, E_h)$ and $G_v = (V, E_v)$ where V is a set of vertices to represent the blocks. On the same layer, an edge must exist between the vertices in either G_h or G_v. For each pair of blocks located on different layers, they may or may not have topological relationship with each other. Thus, an edge may exist between them in either G_h or G_v, but there can also be no such edge.

Alike TCG, LTCG have three properties:

1. G_h and G_v are acyclic.
2. Each pair of nodes i and j, where i and j are assigned to the same layer, must be connected by exactly one edge in G_h or G_v.
3. Let $G_{hk} = (V_k, E_{hk})$ ($G_{vk} = (V_k, E_{vk})$) where $1 \leq k \leq K$ be a sub-graph of $G_h = (V, E_h)$ ($G_h = (V, E_v)$), such that V_k is the set of vertices representing blocks on layer k, $E_{hk} \subseteq E_h$ and E_{hk} contains only those edges with both end points in V_k. For $1 \leq k \leq K$, the transitive closures of G_{hk} and G_{vk} are equal to themselves respectively.

The realization process can be done by performing a longest path search for each node in G_h and G_v, which can be done in $O(n^2)$ time. An example of using LTCG to represent a layered floorplan is shown in Figure 6.1.

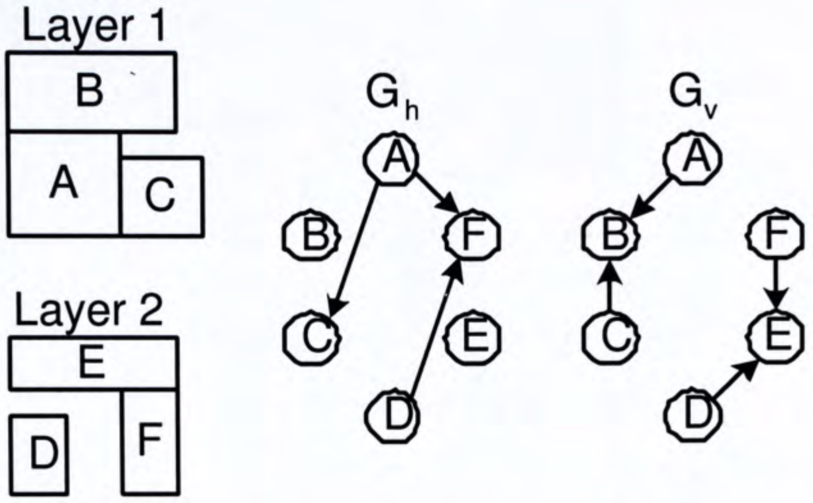


Figure 6.1: A Layered Floorplan and Its LTCG Representation.

6.3.4 Aligning Blocks

There are two graphs, G_h and G_v, in LTCG to govern the horizontal and vertical relationships between the blocks. If we want two blocks X and Y located on different layers to align in the z-direction, the blocks have to overlap vertically and horizontally. To achieve this, a pair of edges (X, Y) an (Y, X), both with zero weight, can be added into the graphs.

For simplicity, we assume that there are only two layers for placing blocks in the following discussion. For a bus to go through a set of blocks, the blocks have to be aligned in such a way that the bus can be routed in a simple geometry. Let $P = \{p_1, p_2, \cdots, p_a\}$ be the set of blocks on the first layer and $Q = \{q_1, q_2, \cdots, q_b\}$ be the set of blocks on the other layer, that a bus goes through. We assume that bus routing on a single layer can be done successfully. Our task is to find a block p_i from P and a block q_j from Q such that p_i and q_i align in the z-direction. In some cases, it is not possible to do alignment for all the buses. As shown in Figure 6.2, block A, D and block B, C cannot be aligned simultaneously and one of the bus has to be considered as infeasible. We have to select a pair of blocks for each bus (if the bus has to go through blocks on two different layers) such that the number of aligned bus is maximized. If

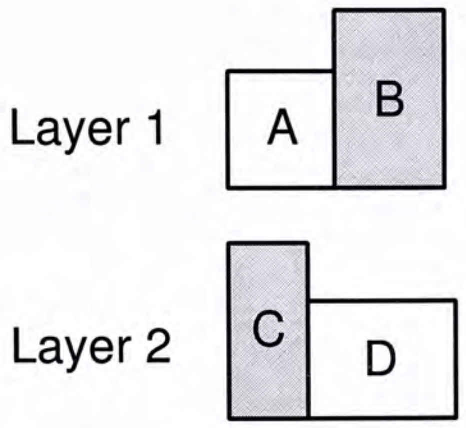


Figure 6.2: Block A, D and Block B, C cannot be aligned simultaneously.

there are K layers for placing blocks, where $K > 2$, a bus with its blocks on k layers where $2 < k \leq K$ can actually be considered as having $k - 1$ sub-buses and the same approach can be applied.

To select pairs of blocks on the same bus to be aligned, we will scan one of the two graphs G_h or G_v first. In our floorplanner, we will consider the graph G_h first. Our proposed method is consisted of several iterations. In each iteration, vertices in G_h that have no incoming edges and participate in no bus are first removed. Then a set S of candidate vertices are found, where each of them has no incoming edges in G_h and belongs to some buses. Then for each bus i, a pair of vertices in S which are on different layers and belong to bus i are matched. After matching, the matched vertices are removed from G_h. If no matching can be done, a vertex is randomly selected from S and removed from G_h. The whole process is repeated (next iteration) until for each bus, one pair of vertices are matched, or until G_h is empty.

Note that matching candidate vertices in a topological order as described above can avoid creating positive cycles in G_h. An example is shown in Figure 6.3. Suppose block B and E are on different layers and belong to bus i, and block C and D are on different layers and belong to bus j. Adding pairs of edges between B, E and C, D simultaneously will yield a positive cycle. In

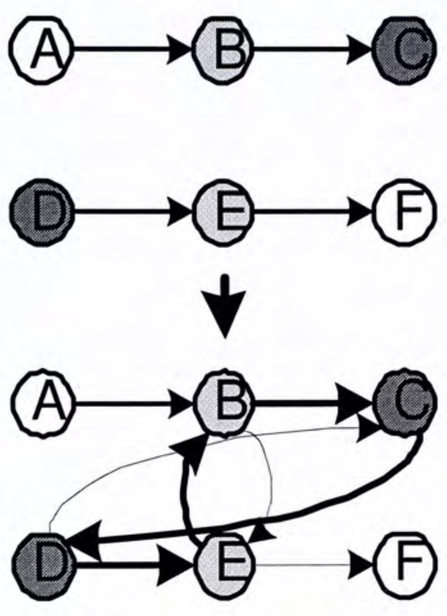


Figure 6.3: Cycle Exists if The Two Pair of Edges are Added Simultaneously.

our approach, we will only match B, E or C, D depending on whether B or D is randomly selected from S and deleted from G_h.

Suppose two blocks X and Y are selected to be matched for bus i. It is guaranteed that no positive cycle will be produced in G_h because none of the two blocks is predecessor of the other in G_h. However, we also need to add those pair of edges (X, Y) and (Y, X) in G_v and cycle may be formed in G_v. If adding a pair of edges to G_v produces a positive cycle, the pair will not be matched and the edges will not be added. At the end, a penalty will be added to the cost function for every unaligned bus. The pseudo code of the procedure to align buses is shown in Figure 6.4.

6.3.5 Solution Perturbation

As mentioned before, simulated annealing is adopted. To change from one candidate solution to another, we have defined several moves: rotate, swap, move, reverse, remove, add, and change-layer. Details of each move will be discussed in the following.

```
ALIGN_BLOCK
align ← 0
FOR i from 0 to number of bus
    matched[i] ← FALSE
ENDFOR
WHILE ($G_h$ not empty) & (align < number of bus)/*start iteration*/
    Remove vertices with no incoming edge and not in any bus
    $S$ ← vertices with no incoming edge
    FOR i from 0 to number of bus
        FOR all pairs $x$, $y \in S$ in bus i and on different layers
            IF matched[i] = FALSE
                IF adding ($x$, $y$) & ($y$, $x$) yield no cycle in $G_v$
                    add ($x$, $y$) & ($y$, $x$) in $G_h$, $G_v$ with weight 0
                    align ++
                    delete vertex $x$ and $y$ in $G_h$
                    $S$ ← $S$ - $\{x,y\}$
                    matched[i] ← TRUE
                END IF
            END IF
        END FOR
    END FOR
    IF no matching is done in this iteration
        $k$ ← randomly select a vertex in $S$
        delete vertex $k$ in $G_h$
    END IF
END WHILE
```

Figure 6.4: Pseudo Code of Aligning Blocks.

Rotate

In this operation, a randomly selected module is rotated. Rotating a module means interchanging the width and height of a module.

Swap

In this operation, two randomly selected modules are swapped. To swap two nodes x and y, exchange the nodes in both G_h and G_v.

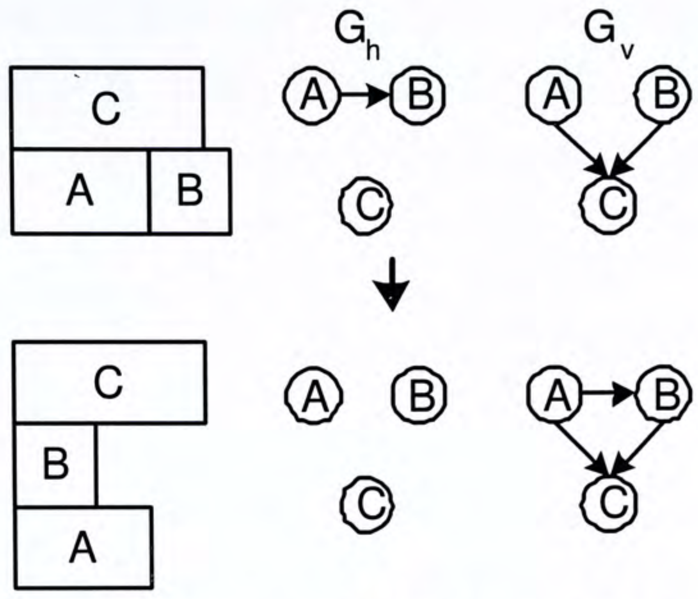


Figure 6.5: A Floorplan Before And After Applying "Move" to Edge (A, B) in G_h.

Move

To "Move" an edge means moving it from either G_h or G_v to the other graph. A reduction edge (x, y) is first selected randomly from G_h or G_v, where x and y are on the same layer. Then, the edge is moved to the other graph. This will change the relationship between x and y from horizontal (vertical) to vertical (horizontal).

To maintain the properties of LTCG, some checkings have to be performed after the move. Assume that (x, y) is moved from G to G'. After moving, for each node $n_i \in F_{in}(x) \cup \{x\}$ in G' and $n_j \in F_{out}(y) \cup \{y\}$ in G', check whether (n_i, n_j) exists in G'. If the edge (n_i, n_j) does not exist, add it to G' and delete the corresponding edge in G. An example of applying "Move" is illustrated in Figure 6.5.

Note that after applying "Move", no cycle will be created. It can be proved by contradiction. Assuming that cycle exists after adding (x, y) to G'. That cycle must involve the edge (x, y) as the original graph is acyclic. This means that there exists a path from y to x in G'. However, according to property 3 of LTCG, if a path exists from y to x in G', an edge $(y,\ x)$ will also exist in G', contradicting to the fact that $(x,\ y)$ is an edge in G. Therefore, after the "Move" operation, the graphs will remain acyclic, and the transitive closure property will also be preserved.

Reverse

Reverse means reversing the direction of a randomly selected reduction edge $(x,\ y)$ in G_h or G_v, where x and y have to be on the same layer. This operation will also change the geometric relationship between the blocks. To reverse an edge $(x,\ y)$ in $G = G_h$ or G_v, delete it from G first and then add $(y,\ x)$ back to G.

Similar to the "Move" operation, checkings have to be performed to maintain the properties of LTCG. For each node $n_i \in F_{in}(y) \cup \{y\}$ in G and $n_j \in F_{out}(x) \cup \{x\}$ in G, check whether $(n_i,\ n_j)$ exists. If $(n_i,\ n_j)$ does not exist, add it to G and delete the corresponding edge in the other graph (G_v or G_h). An example of applying "Reverse" is illustrated in Figure 6.6.

After reversing an edge, the acyclic property and the transitive closure property will also be preserved. The latter is maintained by performing the checkings described above. The former can be proved by contradiction. Assume that cycle exists after reversing an edge $(x,\ y)$. It means that a path exists from x to y originally. This contradicts to the fact that $(x,\ y)$ is a reduction edge in G (there exists no other path from x to y). Therefore, it is guaranteed that the properties of LTCG are maintained after the move.

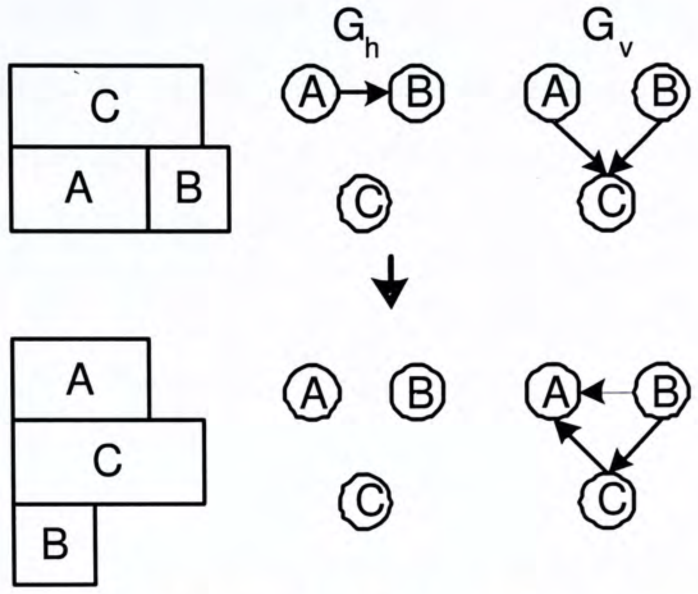


Figure 6.6: A Floorplan Before And After Applying "Reverse" to Edge (A, C) in G_v.

Remove

In LTCG, blocks on different layers may bear one or no relationship with each other. In this operation, the relationship between a pair of randomly selected blocks on different layers is removed.

Add

It is the opposite of "Remove". The "Add" operation adds an edge between a pair of randomly selected blocks in G_h or G_v, where the blocks belong to different layers. Note that after adding the edge, the graph should remain acyclic. If adding a selected edge (x, y) will yield a cycle, the edge will not be added.

Change-Layer

In this operation, a randomly selected block x is moved from one layer to another. It will be placed on the boundary of the destination layer. For every block y that is no longer on the same layer as x, both (x, y) and (y, x) will be removed in both graphs. For every block z that is now on the same layer as x, edges (x, z) are added in C_h as x is placed on the leftmost boundary. Similarly, if x is selected to be placed on the rightmost boundary (the bottomost boundary or the topmost boundary) corresponding edges have to be added in the closure graphs.

6.4 Simulated Annealing

Our objective is to minimize the area of the floorplan and the number of unaligned bus. Thus, the cost function is defined as follows:

$$Cost = \alpha \cdot A + \beta \cdot I$$

where A is the chip area and I is the number of infeasible bus. α and β are parameters that can be specified by the users. Though we aim at minimizing the area of the floorplan and the number of infeasible buses, other aspects like total wire length and routing congestion can be taken into account by including more terms in the cost function.

6.5 Soft Block Adjustment

After placing the modules as hard blocks, soft block adjustment is done to change the shapes of the blocks to make the resultant floorplan more compact. This is again done by simulated annealing. In each perturbation, a block is selected randomly. The shape of it is changed a little bit, as long as it does not violate the aspect ratio of the block. The cost function is defined as follows:

$$\text{cost} = \alpha\cdot\ A + \beta\cdot\ S$$

where A is the total area of the floorplan and S is the difference between the preset aspect ratio bound and the actual aspect ratio bound of the floorplan. This step is shown to be essential by the experimental results as it can greatly reduce the deadspace of the floorplan.

Benchmark	No. of Blocks	No. of Layers
ami33	33	4
ami49	49	4

Table 6.1: Characteristics of Data Set 1.

Benchmark	Layer	Dead Space ([4])	Dead Space (LTCG)	Time (s) (LTCG)
ami33	4	3.09%	1.95%	13
ami49	4	3.76%	2.49%	28

Table 6.2: Comparisons between [4] and LTCG.

6.6 Experimental Results

A 3D floorplanner was implemented with the C++ language, using the LTCG representation. All the experiments were conducted in an Intel P4 (3.2 GHz) machine with 2G memory. The MCNC benchmarks and the GSRC benchmarks were used. We have conducted two sets of experiments. The first set of experiments does not consider buses. The characteristics of the benchmarks (data set 1) are shown in Table 6.1. The results are shown in Table 6.2. Comparisons showed that our floorplanner outperforms the floorplanner proposed in [4]. As the runtimes were not reported in [4], only the runtimes of our floorplanner are shown in Table 6.2. For all the experiments, the best of twenty trials are reported. The runtime reported is the average of the twenty trials.

The second set of experiments considers buses. Buses are randomly constructed in the benchmarks. We have constructed two sets of data (data set 2 and data set 3). The characteristics of data set 2 are shown in Table 6.3. The number of buses involved is large, though the number of blocks a bus goes through is small. The characteristics of data set 3 are shown in Table 6.4. The number of buses involved is smaller, but the number of blocks involved in each bus is huge. The experimental results after soft block adjustment are shown

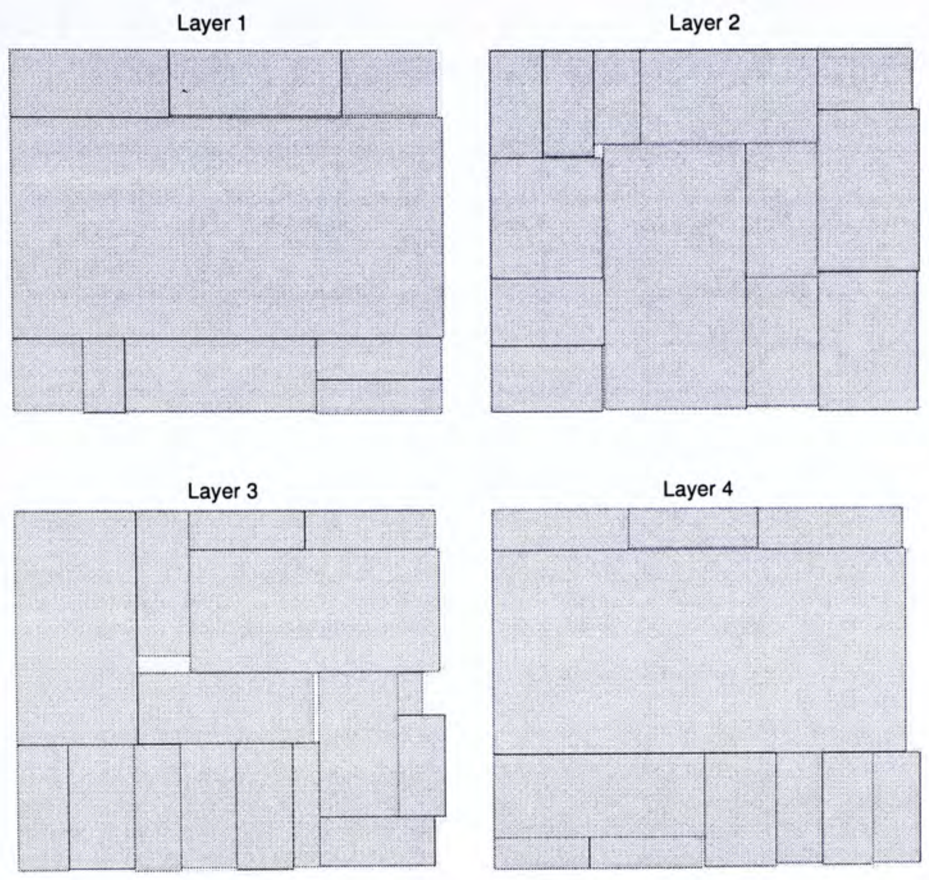


Figure 6.7: Result of ami49 in Data Set 1.

in Table 6.5 and Table 6.6. All floorplans are packed successfully with every inter-layer bus aligned. The deadspace is 5.86% on average for data set 2, and 4.97% on average for data set 3. Experimental results showed that LTCG is very promising for multi-layer floorplanning and can handle inter-layer block alignment very effectively.

6.7 Summary

As the complexity of VLSI circuit design increases, the number of interconnects involved has grown rapidly. 3D chips can reduce interconnect lengths significantly. However, there was not much work done in 3D floorplanning. It is a problem yet to be explored. In this chapter, we have propsoed a 3D floorplan representation namely *Layered Transitively Closure Graph* (LTCG), based on the Transitive Closure Graph [32] representation for non-slicing floorplans. Besides a pair of graphs G_h and G_v, LTCG also stores the layer information

Benchmark	No. of Blocks	No. of Layers	No. of Buses	Average/Max. No. of Blocks in a Bus
apte	9	2	3	2/2
hp	10	2	3	2/2
xerox	11	2	3	2/2
ami33	33	3	7	2.29/3
ami49	49	4	7	2.29/3
n100a	100	4	10	2.21/3

Table 6.3: Characteristics of Data Set 2.

Benchmark	No. of Blocks	No. of Layers	No. of Buses	Average/Max. No. of Blocks in a Bus
apte	9	2	2	4/4
hp	10	2	1	6/6
xerox	11	2	2	4.5/5
ami33	33	3	2	7.5/8
ami49	49	4	3	7/9
n100a	100	4	5	7.4/10

Table 6.4: Characteristics of Data Set 3.

of each block. Based on LTCG, we proposed a method to align blocks on different layers, by adding pair of edges in LTCG. A floorplanner was implemented using the LTCG representation, and the experimental results are very promising.

6.8 Acknowledgement

We would like to thanks Royce L.S.Ching for helping in extending our bus alignment algorithm from 2D to 3D.

Benchmark	Time(s)	Deadspace
apte	2	2.06%
hp	4	8.18%
xerox	4	5.47%
ami33	23	5.32%
ami49	89	6.30%
n100a	371	7.84%

Table 6.5: Experimental Results of Data Set 2.

Benchmark	Time(s)	Deadspace
apte	4	1.77%
hp	3	8.44%
xerox	7	3.16%
ami33	28	5.24 %
ami49	63	4.10%
n100a	545	7.13%

Table 6.6: Experimental Results of Data Set 3.

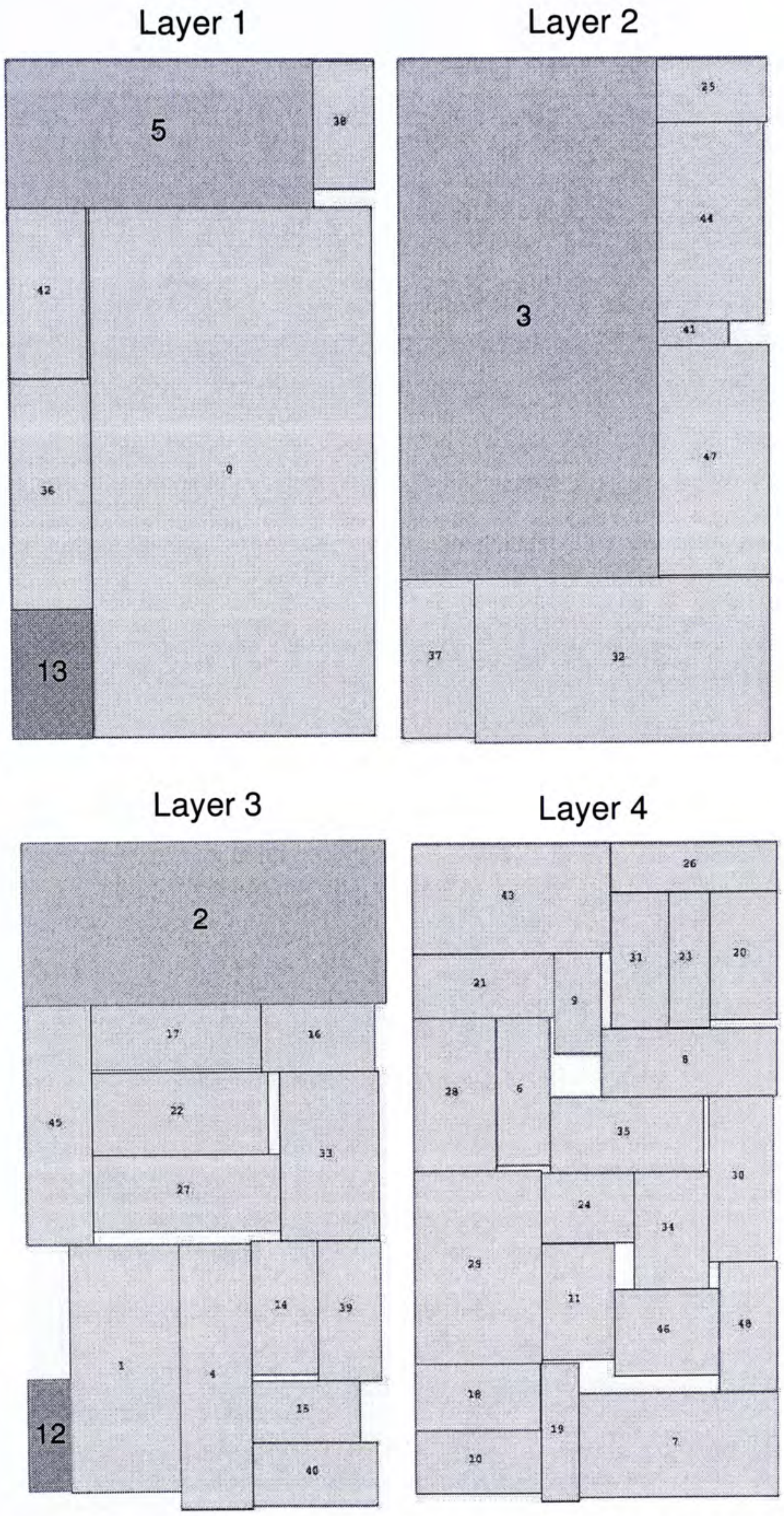


Figure 6.8: Result of ami49 in Data set 3 ($B_1 = \{0\text{-}5, 32, 33, 44\}$, $B_2 = \{6\text{-}11\}$, $B_3 = \{12\text{-}17\}$).

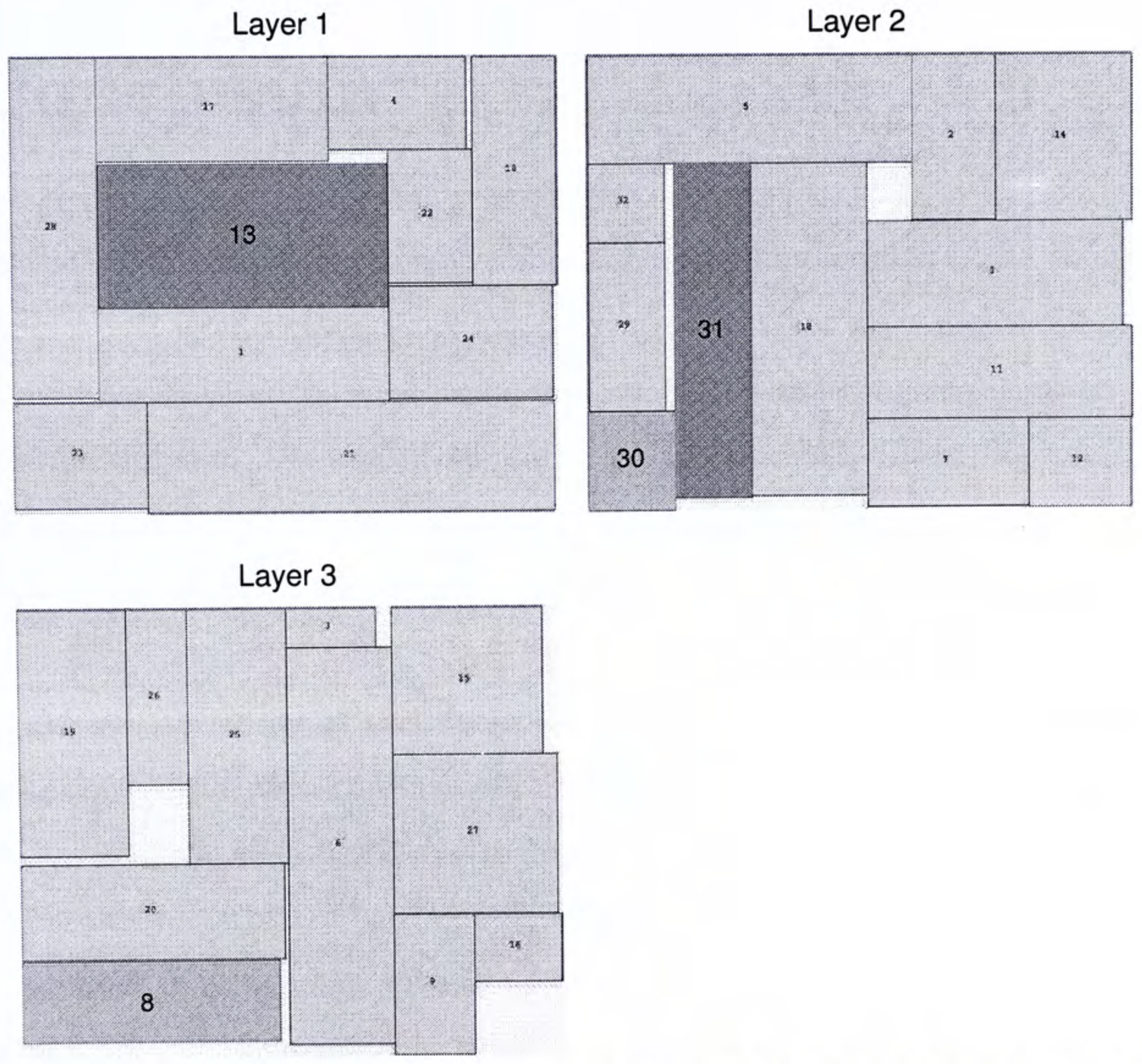


Figure 6.9: Result of ami33 in Data set 3 ($B_1 = \{6\text{-}9, 26, 29, 30\}$, $B_2 = \{10\text{-}14, 18, 24, 31\}$).

Chapter 7

Conclusion

As the VLSI technology advances into the deep submicron era, the complexity of VLSI circuit design has increased greatly. Not only has the number of modules involved become large, the number of interconnects involved was also multiplying. We are interested in the interconnect-driven floorplanning problem as it is an important issue in floorplanning in this deep submicron era. Obtaining a compact floorplan is not enough, it is the routability that matters.

At the beginning of this thesis, an overview of the VLSI design cycle is presented. After that, an introduction to the physical design cycle is given. In our research, we focused on the floorplanning phase. We have reviewed the literatures on 2D floorplan representation, 3D floorplan representation, and bus-driven floorplanning. We have proposed an algorithm to solve the bus-driven floorplanning problem in 2D floorplan. We have also proposed a 3D floorplan representation.

Bus-driven floorplanning is a floorplanning problem with bus planning taken into consideration. Bus is a collection of wires running over a set of modules. To solve the bus-driven floorplanning problem in 2D floorplans, we use the sequence pair representation for general non-slicnig floorplans. The input of the problem is a set of blocks and a set of buses, where each bus has to go through a set of blocks. We have to decide the position of each block and each bus such that all the buses are just 0-bend, 1-bend, or 2-bend. We have derived some necessary conditions for a bus to go through its blocks. Simulated annealing is adopted to find a solution, and a floorplan is evaluate according to the packing area, the total bus area, and the number of infeasible buses. Experimental results have demonstrated the strength of our proposed algorithm.

As the complexity of VLSI circuit design increases, the number of interconnects involved has grown rapidly. 3D chips can reduce interconnect lengths significantly. However, there was not enough EDA tools for 3D circuit design, and there was not much work done in 3D floorplanning. It is a problem yet to be explored. We have proposed a 3D floorplan representation namely *Layered Transitive Closure Graph (LTCG).* It is based on the Transitive Closure Graph [32] representation for non-slicing floorplans. Beside a pair of graphs G_h and G_v, LTCG also stores the layer information of each block. Based on LTCG, we proposed a method to align blocks (of the same bus) on different layers, by adding pairs of edges in LTCG. A 3D floorplanner was implemented using the LTCG representation, and the experimental results is very promising.

Bibliography

[1] H. Xiang, X. Tang, and M. D.F.Wong, Bus-Driven Floorplanning, in *International Conference on Computer Aided Design*, 2003.

[2] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Massachusetts 02061, USA, third edition edition, 1999.

[3] J. Cong, Challenges and Opportunities for Design Innovations in Nanometer Technologies, in *SRC Design Sciences Concept Paper*, 1997.

[4] J. Berntsson and M. Tang, A Slicing Structure Representation for the Multi-Layer Floorplan Layout Problem, in *European Workshop on Evolutionary Computation in Hardware Optimization*, 2004.

[5] G. E. Moore, Gramming More Components onto Integrated Circuits, in *Electronics*, volume 38, 1965.

[6] S. Sutanthavibul and Rosen., An Analytical Approach to Floorplan Design and Optimization, in *IEEE Transaction on Computer-Aided Design*, pages 761–769, 1991.

[7] T. Chen and M. K. H. Fan, On Convex Formulation of the Floorplan Area Minimization Problem, in *Proceedings of the 1998 International Symposium on Physical Design*, pages 124–128, 1998.

[8] N. Metropolis, A. Rosenbluth, M. N. Rosenbluth, A. Teller, and E. Teller, Equations of State Calculations by Fast Computing Machines, in *J. Chem. Phys 21*, pages 1087–1092, 1958.

[9] M. Pincus, A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems, in *Oper. Res. 18*, pages 1225–1228, 1970.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by Simulated Annealing, in *Science*, volume 220, pages 671–680, 1983.

[11] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, Rectangle-Packing-Based Module Placement, in *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, pages 472–479, 1995.

[12] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, B*-Trees: A New Representation for Non-Slicing Floorplans, in *Proceedings of the 37th Conference on Design Automation*, pages 458–463, 2000.

[13] E. F.Y.Young, C. C.N.Chu, and C. Shen, Twin Binary Sequences: A Non-Redundant Representation for General Non-Slicing Floorplan, in *Proceedings of the 2002 International Symposium on Physical Design*, pages 196–201, 2002.

[14] C. W. Sham and E. F. Y. Young, Routability Driven Floorplanner with Buffer Block Planning, in *Proceedings of the International Symposium on Physical Design*, pages 50–55, 2002.

[15] K. K. C. Wong and E. F. Y. Young, Fast Buffer Planning and Congestion Optimization in Interconnect-Driven Floorplanning, in *Proceedings of the conference on Asia South Pacific Design Automation Conference*, pages 411–416, 2003.

[16] J. H. Holland, Adaptation in Natural and Artificial Systems, in *Ann Arbor*, MI: The University of Michigan Press, 1975.

[17] M. Rebaudengo and M. Reorda, Gallo: A Genetic Algorithm for Floorplan Area Optimization, in *IEEE Transaction on Computer-Aided Design*, volume 15, pages 943–951, 1996.

[18] X. Hong et al., Corner Block List: An Effective and Efficient Topological Representation of Non-slicing Floorplan, in *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design*, pages 8–12, 2000.

[19] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, Module Placement on BSG-Structure and IC Layout Applications, in *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, pages 484–491, 1996.

[20] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, An O-tree Representation of Non-slicing Floorplan and Its Applications, in *Proceedings of the 36th ACM/IEEE Conference on Design Automation*, pages 268–273, 1999.

[21] P. Pan and C. L. Liu, Area Minimization for Floorplans, in *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, volume 14, pages 123–132, 1995.

[22] F. Y. Young, C. C. N. Chu, W. S. Luk, and Y. C. Wong, Floorplan Area Minimization Using Lagrangian Relaxation, in *Proceedings of the 2000 International Symposium on Physical Design*, pages 174–179, 2000.

[23] J. H. Y. Law and E. F. Y. Young, Multi-Bend Bus Driven Floorplanning, in *Proceedings of the 2005 International Symposium on Physical Design*, pages 113–120, 2005.

[24] R. H. Otten, Automatic Floorplan Design, in *Proceedings of the 19th conference on Design automation*, pages 261–267, 1982.

[25] D.F.Wong and C.L.Liu, A New Algorithm for Floorplan Design, in *Proceedings of the 23rd ACM/IEEE conference on Design automation*, pages 101–107, 1986.

[26] W. Shi, An Optimal Algorithm for Area Minimization of Slicing Floorplans, in *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, pages 480–484, 1995.

[27] H. Onodera, Y. Taniguchi, and K. Tamaru, Branch-and-Bound Placement for Building Block Layout, in *Proceedings of the 28th Conference on ACM/IEEE Design Automation*, pages 433–439, 1991.

[28] T.-C. Wang and D.F.Wong, An Optimal Algorithm for Floorplan Area Optimization, in *Proceedings of the 27th ACM/IEEE Conference on Design Automation*, pages 180–186, 1990.

[29] Y. Pang, C.-K. Cheng, and T. Yoshimura, An Enhanced Perturbing Algorithm for Floorplan Design Using the O-Tree Representation, in *Proceedings of the 2000 International Symposium on Physical Design*, pages 168–173, 2000.

[30] B. Yao, H. Chen, C.-K. Cheng, and R. Graham, Revisiting Floorplan Representations, in *Proceedings of the 2001 International Symposium on Physical Design*, pages 138–143, 2001.

[31] K. Sakanushi and Y. Kajitani, The Quarter-State Sequence (Q-sequence) to Represent the Floorplan and Applications to Layout Optimization, in *Proceedings of the 2002 IEEE Asia-Pacific Conference on Circuits and Systems*, pages 829–832, 2000.

[32] J.-M. Lin and Y.-W. Chang, TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans, in *Proceedings of the 38th Conference on Design Automation*, pages 764–769, 2001.

[33] X. Tang and D.F.Wong, FAST-SP: A Fast Algorithm for Block Placement Based on Sequence Pair, in *Proceedings of the 2001 Conference on Asia South Pacific Design Automation*, pages 521–526, 2001.

[34] Y. Deng and W. P. Maly, Interconnect Characteristics of 2.5-D System Integration Scheme, in *Proceedings of the 2001 International Symposium on Physical Design*, pages 171–175, 2001.

[35] P. H. Shiu, R. Ravichandran, S. Easwar, and S. K. Lim, Multi-Layer Floorplanning for Reliable System-on-Package, in *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems*, 2004.

[36] J. Cong, J. Wei, and Y. Zhang, A Thermal-Driven Floorplanning Algorithm, in *Proceedings of the 2004 International Conference on Computer Aided Design*, 2004.

[37] J. Xu, P. N. Guo, and C. K. Cheng, Rectilinear Block Placement Using Sequence-Pair, in *Proceedings of the 1998 International Symposium on Physical Design*, pages 173–178, 1998.

[38] Y. Ma et al., Floorplanning with Abutment Constraints and L-Shaped/T-Shaped Blocks Based on Corner Block List, in *Annual ACM IEEE Design Automation Conference*, 2001.

[39] E. F.Y.Young, C. C.N.Chu, and M.L.Ho, A Unified Method to Handle Different Kinds of Placement Constraints in Floorplan Design, in *Proceedings of the 2002 with EDA Technofair Design Automation Conference Asia and South Pacific*, 2002.

[40] X. Tang and D.F.Wong, Floorplanning with Alignment and Performance Constraint, in *Annual ACM IEEE Design Automation Conference*, 2002.

[41] X. Tang and M. D.F.Wong, On Handling Arbitrary Rectilinear Shape Constraint, in *Proceedings of the 2004 with EDA Technofair Design Automation Conference Asia and South Pacific*, pages 38–41, 2004.

[42] T.-C. Chen and Y.-W. Chang, Modern Floorplanning Based on Fast Simulated Annealing, in *Proceedings of the 2005 International Symposium on Physical Design*, pages 104–112, 2005.

[43] J. Parkhurst, N. Sherwani, S. Maturi, D. Ahrams, and E. Chiprout, SRC Physical Design Top Ten Problems, in *Proceedings of the 1999 International Symposium on Physical Design*, pages 55–58, 1999.