Fixed-Outline Bus-Driven Floorplanning

JIANG, Yan

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
August 2011

Thesis/Assessment Committee

Professor WU Yu Liang (Chair)
Professor YOUNG Fung Yu (Supervisor)
Professor LEE Ho Man (Committee Member)
Professor WONG Ngai (External Examiner)

Abstract

Bus routing has become an important research topic as the interconnection of chips becomes increasingly congested. Bus is a collection of wires that carry signals between different blocks. In order to ease bus routing and avoid unnecessary iterations of the physical design cycle, it would be favourable to consider bus routing in a floorplanner that minimizes chip area, interconnect cost and bus length. Therefore bus-driven floorplanning becomes an important topic in VLSI design. It considers bus assignment during floorplanning. In many previous works [1] [2] [3] [4] [5], bus-driven floorplanning is mostly aimed at minimizing the chip area and total bus length.

In this thesis, we solve the fixed-outline bus-driven floorplanning problem with an objective to minimize the total bus length and interconnect cost. In the first approach, we focus on the bus length minimization problem. Insteading of adding up all the bus length in the cost function of the simulated annealing process, we handle the bus length minimization problem globally in a more sophisticated way by using a bus-driven partitioning step. In this step, min-cut partitioning is performed recursively to reduce bus length. At the same time, we use a more accurate estimation algorithm that connects buses to the block boundaries to estimate bus length. We can achieve shorter bus length on average by using this approach. Experimental results show that we can improve over the most closely related previous work [4] in terms of bus length, running time and dead space. Besides, we also

improve over a most recent work [5] in terms of bus length on average.

In the second approach, we focus on the bus routability problem. We handle the bus routability problem in a more sophisticated manner to increase the searching space and give more flexibility by changing the bus shape and the positions of the bus bendings. We present a floorplanner that can give a fixed-outline floorplanning solution including bus route that minimizes bus length, number of extra vias and interconnect cost. Our goal is to route all the bus components successfully in each iteration of the simulated annealing process of this floorplanner. L shape bus component is used to route the buses with conflicts with other bus components. Experimental results show that we also improve over the most recent work [5] in terms of bus length and successful rate on average. Besides, we can improve over our first floorplanner in Chapter 3 and the modified floorplanner of [4] in terms of both bus length, running time and successful rate.

隨著芯片的連接變得越來越密集，總線的佈線成爲了一個重要的研究課題. 總線是一組需要經過數個組件的電線傳輸著電子信號. 爲了簡化總線佈線和避免重複不必要的物理設計流程 設計一個考慮總線佈線同時最小化芯片面積,連接代價和總線長度的布圖工具是很好的選擇. 因此，總線主導的布圖規劃成爲了超大規模集成電路設計技術中的一個重要研究課題. 在以往所發表的研究工作中，總線主導的布圖規劃主要側重芯片面積和總線長度.

在這篇論文中,我們關注考慮限定外形同時最小化總線長度和連接代價的總線主導的布圖規劃問題. 在第一個方法中，我們側重於總線長度的最小化問題. 不同於其他發表的研究工作，我們以全局觀處理總線的最小化. 採用總線主導的分區方法通過尋找最小割集來減少總線長度. 同時,我們採用更加精確的算法估算總線長度 . 通過這個方法,我們可以獲得相當短的總線長度. 實驗結果表明和以往的研究工作比較我們可以減少總線長度,運行時間以及空白區域的比例.

在第二個方法中,我們關注總線可繞性問題 . 在以往發表的研究工作中，對於模擬退火過程中有不可繞的總線的解，研究者會用一個很簡單的方法處理，比如在代價函數中加一個懲罰. 不同於以往的研究工作，爲了擴大搜索空間增加總線形狀以及屈曲位置的靈活度，我們採用一種更爲複雜的方法處理這種情況 我們研發了一個考慮限定外形同時最小化總線長度，額外焊接孔的數目和連接代價的布圖工具. 我們的目標是爲每一個模擬退火迭代找到一個所有總線可繞的解 採用一個新的名爲L 型總線分量的模型來爲有衝突約束條件的總線分量佈線. 實驗結果顯示我們可以比以前的研究工作工作有更短的平均總線長度和更高的平均成功率. 同時這個方法比我們的第一個方法有更短的總線長度和運行時間以及更高的成功率.

Acknowledgement

I would like to thank my supervisor, Professor Evangeline F.Y. Young. She is a very nice and excellent professor and supervisor. In the last two years, she gave me a lot of good advices and encouragements. She encouraged me when I felt disappointed. She gave me directions when I felt lost. I must not be able to finish my research without the effort she put on me.

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

Finally, I would like to thank all my colleagues in the CAD group. I really enjoy the life that we learn, share and research together in Room 506.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The internet has emerged as a medium to distribute information, communicate, plan event and conduct E-commerce. *Integrated Circuits (IC)* technology is important in this information revolution because ICs are widely used in devices that changes our perspective of work, life at home and provides new tools for entertainment. ICs are used in microprocessor for computers, memory and interface chips. ICs are also used in computer networking, switching systems, communication systems, cars, airplanes and microwave ovens. ICs are even used in toys, hearing aids and implants for human.

The widespread use of ICs is due to the revolution in IC technology. In the 1960s, the IC technology has evolved from the integration of a few transistors to the integration of millions of transistors in *Very Large Scale Integration (VLSI)* chips today. According to Moore's Law [6], it was predicted that the number of transistors in a single IC will double in every 1.5 years. As long as a transistor becomes smaller, it becomes faster, conducts more electricity, and uses less power. The cost of producing each transistor goes down, and more of them can be packed on a chip. The technology of IC continues to scale down, to produce faster, more complicated yet more powerful ICs.

With the rapid growth in the integration technology, it is necessary to bring in automation in the design of VLSI chips. As technology continues to scale down, and the IC designer can hardly transform a circuit description into a *layout* all manually. VLSI physical design automation provides efficient algorithms and data structures to find good arrangements of devices or efficient interconnect schemes which play a key role in improving the performance of a chip. In addition, efficient algorithms not only lead to fast turn-around time, but also permit designers to make iterative improvements to the layouts.

In the following sections, the physical design cycle will be described briefly. After that, the floorplanning problem will be introduced and discussed.

1.1 Physical Design

Physical design is the process to transform a circuit description of an IC into a geometric description (layout). Layout is created by converting each logic component (cells, macros, gates, transistors) into a geometric representation (specific shapes in multiple layers). Physical design is a very complicated process and therefore it is usually broken down into several stages, such as partitioning, floorplanning, placement, routing and compaction. The different stages of the physical design cycle are shown in Figure 1.1. The details of each stage will be discussed in this section.

Partitioning

In the partitioning stage, the whole circuit is decomposed into several finer subcircuits called *blocks*. The partitioning stage is necessary because a chip may actually be comprised of millions of transistors in order to achieve complicated functionalities. Huge circuits are hard to be managed efficiently and cannot be

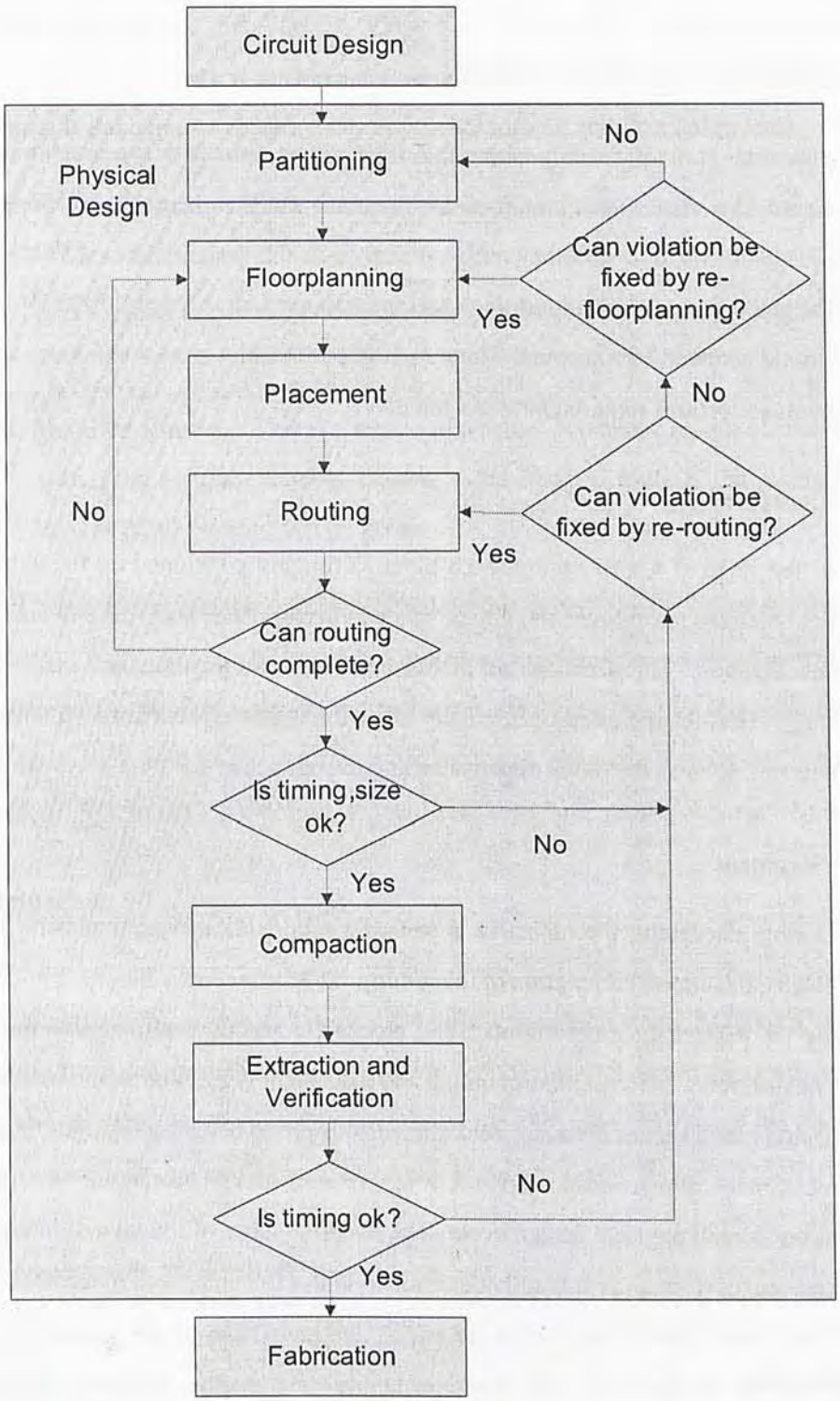

Figure 1.1: Physical design cycle

placed all at once, decomposition into finer sub-systems is thus a must in the design cycle. After partitioning, each sub-circuit 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. The output of partitioning is a set of blocks and the interconnections required between them.

Floorplanning

In the stage of floorplanning, each block is roughly positioned on the chip with the objective to optimize the circuit size and performance according to the circuit specification. The decisions on block shapes and pin positions are made in this stage. Besides the circuit size, many other important aspects are also taken into account, such as the block dimensions and overall delay.

Placement

During placement, the objective is to find a minimum area arrangement for the blocks that allows completion of interconnections between the blocks, while meeting the performance constraints. The blocks are exactly positioned on the chip. The difference between floorplanning and placement is that floorplanning needs to decide block shapes because block dimensions are to be determined but in placement, block dimensions are known. Floorplanning and placement are very crucial in the overall physical design cycle. The overall quality of the layout, in terms of area and performance is mainly determined in floorplanning and placement.

Routing

During routing, the interconnections between blocks are completed according to the specified netlist. A router needs to complete all circuit connections on the

active layers and the metal layers. On the active layers, the routing space, i.e., the space not occupied by the blocks is partitioned into channels and switchboxes. Routing is done in two phases. They are global routing and detailed routing.

1. **Global Routing:** This is a step to plan different routes from a global point of view. In this step, a list of routing regions is assigned to each net without specifying the actual routing layout. It is a rough plan to check whether completion of all interconnections is possible.

2. **Detailed Routing:** This is a step to find actual geometric layout of each net within the assigned routing regions. After detailed routing, the geometric layouts of all the nets will be known.

If some of the connections are not able to be routed, the *rip-up and re-route* technique will be used. Some of the routed connections are removed and then re-routed 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

After the routing stage, the layout is functionally completed. To reduce cost, the layout needs to be as small as possible. In the compaction stage, a layout is compressed from different directions in order to reduce the total area without violating any design rules. By minimizing the area of the chip, one can reduce the cost of manufacturing as more chips can be produced on one wafer.

Extraction and Verification

In this stage, the layout is verified. The first process is *Design Rule Checking*. All the geometric patterns are verified to ensure that they meet the design rules

imposed by the fabrication process. After checking the layout for design rule violations and removing the design rule violations, the functionality of the layout is verified by *Circuit Extraction* that generates a circuit representation from the layout. In the process of *Layout Versus Schematics (LVS) verification*, the extracted description is compared with the original circuit description to verify its correctness. In the process of *Performance Verification*, accurate calculation of the timing of each component, including interconnection is done based on the values of the resistances and capacitances that are computed from the extracted geometric information. In the process of *Reliability Verification*, the reliability aspects of the layout are checked based on the extracted information to ensure that the layout will not fail due to electro-migration, self-heating and other effects. If any problem is found, engineers may need to go back to the earlier designing steps to fix the problem.

1.2 Floorplanning

Floorplanning is a crucial step in the early design phase because good planning can avoid unnecessary iterations in the design cycle. In the floorplanning phase, the position and shape of each block are planned to optimize circuit performance. The input to this 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 the blocks. There are two kinds of blocks. Some blocks' shapes are known and cannot be changed. They are called *hard blocks*. The other kind of blocks for which shapes are yet to be determined are called *soft blocks*. A formal definition of the floorplanning problem is given as follows:

Given a set of n blocks $\{b_1, b_2, \cdots, b_n\}$, where each block b_i is associated with an area A_i, together with two aspect ratio bounds r_i^l and r_i^h such that $r_i^l \leq h_i/w_i \leq r_i^h$, where h_i and w_i is the height and width of block b_i. The output of the problem is the

x- and y-coordinates and the dimension (h_i, w_i) of each block b_i. There should be no overlapping between blocks, and the circuit performance should be optimized.

To optimize circuit performance, there are some objectives that need to be considered in floorplanning. In this section, some floorplanning objectives and some common approaches adopted today to solve the floorplanning problem will be presented.

1.2.1 Floorplanning Objectives

There are several objectives to be optimized in floorplanning, including the total chip area, the total wire length, the critical path delay etc.

Chip Area

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

Total Wire Length

Another important objective is to minimize the total wire length. To reduce the production cost, the total wire length is minimized so as to use shorter wires to connect blocks. Besides, minimizing the total wire length is good for timing.

Delay

In some cases, minimizing the total wire length is not enough. Timing is an important issue. To optimize the final circuit performance, the delay on the critical path should be minimized.

Routability

Routability refers to the possibility of completing all the connections. A good floorplan solution should be a solution with high routability because 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 power consumption, minimizing heat dissipation, etc. In our work, we focus on the fixed-outline bus-driven floorplanning problem by arranging the blocks on the same bus in such a way that bus routing can be done effectively.

1.2.2 Common Approaches

Much work has been done on floorplan representations. Floorplans can be classified into three main categories: slicing [10][11][12], non-slicing [13][14][15][16][17] [18][19], and mosaic [21][23][22][24] in Figure 1.2.

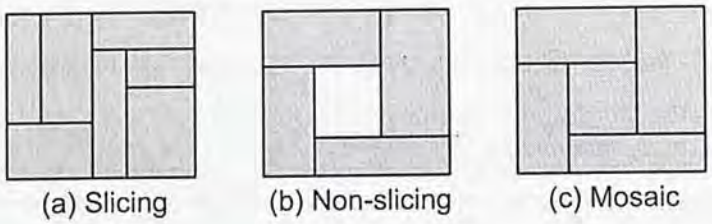

Figure 1.2: Examples of the three main kinds of floorplans.

In slicing structure, a floorplan can be obtained by recursively dividing a rectangle into smaller rectangles using a horizontal or a vertical cut. *normalized Polish expression* [11] is a widely adopted slicing floorplan representation. However,

most of the real designs are not in slicing structure. Thus slicing representation is not general enough.

A *non-slicing* floorplan is a floorplan that is not necessarily slicing (Figure 1.2(b)). It is the most general kind of floorplans. Much work has been done on non-slicing floorplan representation, e.g., *B*-Tree* [13], *O-Tree* [14], *sequence pair* [17], *BSG* [19], and *Transitive Closure Graph* [20].

Mosaic floorplan represents 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 1.2(c)). Each module corner is formed by a T-junction (no +-junction), except those at the four corners of the floorplan.

The floorplanning problem is proved to be NP-complete. Several research explored floorplanning algorithms [25][31], including five categories: (1) constraint based approach [26], (2) integer programming based approach [25], (3) rectangular dualization based approach [27], (4) hierarchical tree based approach [28], (5) iterative approaches [35]. In constraint based approach, a floorplan of optimal area under a given set of horizontal and vertical topological (ordering) constraints. In integer programming based approach, floorplanning problem is modeled as a set of linear equations with 0/1 integer variables. Overlap constraints and routability constraints are considered. In rectangular dualization based approach, floorplanning problem is modeled by converting a partition graph into its rectangular dual. In hierarchical tree based approach, a floorplan is represented by a tree generated by using top-down partitioning or bottom-up clustering method. In iterative approaches, it starts from an initial floorplan solution, and performs a series of perturbations on candidate solution to search a better solution until no more improvement can be achieved. Later on, some timing-driven floorplanning approaches were proposed to solve the performance problems [29][30]. In the following paragraphs, analyti-

cal approach, genetic algorithm and simulated annealing will be described briefly. The first one is an integer programming based approach. The second one and the third one are iterative approaches.

Analytical Approach

In [32], the floorplanning problem is formulated as a *mixed integer linear program (MILP)*, where all the constraints are linear functions, the objective function is a linear function, and the variables are real numbers or integers. However, the run time to solve the MILP is exponential to the number of variables and equations. MILP is also an NP-complete problem. Thus, it is not good to use this approach to solve problems of large scale. In [33], the aspect ratios of the blocks are handled in an indirect way and a convex formulation is proposed to reduce the number of variables and constraints used.

Genetic Algorithm

Genetic algorithm [34][35] is a stochastic searching approach to solve NP-complete problems. It belongs to the larger class of evolutionary algorithms, which generates solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. The process starts with a set of randomly generated initial solutions named population. In each generation, the fitness of every solution in the population is evaluated. Multiple solutions are stochastically selected from the current population based on their fitness, and modified by using two types of genetic operators, mutation and crossover, to form a new population. Mutation means modifying one solution by applying a small change to it. Crossover means forming a new solution by re-combing two solutions in the population. The new population is then used in the next iteration of the algorithm. Usually, the algorithm terminates when either a maximum number

of generations has been produced, or a satisfactory fitness level has been reached for the population. A pseudo code of the general genetic algorithm approach is described in Figure 1.3.

GENETIC_ALGORITHM (P, R_c, R_m)

```
1  X ← {x1, x2, ···, xP}
2  WHILE stopping criteria not met
3      Xnew ← ϕ
4      WHILE number of children created < P×Rc
5          select two solutions xi and xj from X
6          xnew ← crossover(xi, xj)
7          Xnew ← Xnew ∪ {xnew}
8      END WHILE
9      select P solutions from X∪Xnew, and call it X
10     WHILE number of children mutated < P×Rm
11         select a solution xk from X
12         xnew ← mutate(xk)
13         Xnew ← Xnew ∪ {xnew}
14     END WHILE
15     X ← Xnew
16 END WHILE
17 RETURN the best solution in X
```

Figure 1.3: Pseudo code of the genetic algorithm.

Simulated Annealing (SA)

SA is another heuristic searching approach to solve NP-complete problems and it belongs to the probabilistic and iterative class of algorithms. The authors of [36] proposed a SA algorithm to find the equilibrium configuration of a collection of atoms at a given temperature. The authors of [37] then proposed using SA as an optimization tool. After that, the authors of [38] suggested that SA can be used as a general approach for different global optimization problems. It is widely used to solve the floorplanning problem, such as in [17][13][22][39][40].

The name and inspiration of SA come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat excites the atoms from their initial positions (a local minimum of the internal energy) and makes them wander randomly through states of higher energy. The slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

By analogy with this physical process, each iteration of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends both on the difference between the corresponding function values and also on a global parameter t (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when t is large, but increasingly "downhill" as t goes to zero. The allowance for "uphill" moves saves the method from getting stucked at the local optimum.

In a SA based floorplanner, each floorplan solution s represented by a representation (e.g., sequence pair representation, B*-tree, etc.) of the search space is analogous to a state of a physical system, and the cost function f_s to be minimized is analogous to the internal energy of the system. The cost function may take area, wire length, etc. into consideration to evaluate the quality of a candidate floorplan

solution. The goal is to get a floorplan solution with the minimum possible cost starting from an arbitrary initial solution. SA starts with an initial solution s_0, and an initial temperature t_0. In each iteration, the candidate solution is changed a little and is evaluated by the cost function f_s. If the cost of the new solution is less than that of the old one, the new solution is accepted. Otherwise, the new solution is accepted according to a probability depending both on the difference between the cost function values and also on the temperature t. The temperature t will be lowered at a cooling rate c. Finally, the process will terminate when the temperature is below a threshold t_h. The pseudo code is shown in Figure 1.4.

SIMULATED_ANNEALING ($ITER$, t_0, t_h, c)

```
1  s ← s0
2  t ← t0
3  WHILE t ≥ th
4      FOR i from 1 to ITER
5          snew ← move(s)
6          Δf ← cost(snew) - cost(s)
7          r ← random number between 0 to 1
8          IF Δf < 0 OR r < exp(−kΔf/t)
9              s ← snew
10         END IF
11     END FOR
12     t ← t × c
13 END WHILE
14 RETURN s
```

Figure 1.4: Pseudo code of SA.

1.3 Motivations and Contributions

In the deep sub-micron era, the number of transistors and interconnections are growing rapidly in VLSI systems. 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 of having a lot of unroutable buses. In order to ease bus routing and avoid unnecessary iterations in the design cycle, we incorporate bus planning in the early designing phase. This is our motivation to solve the fixed-outline bus-driven floorplanning problem.

In many previous works [1] [2] [3] [4] [5], bus-driven floorplanning aims mostly at minimizing the chip area and the total bus length. However, none of these works considers fixed-outline constraint and wire length. And in most of those works, buses are assumed to be connected to the centers of their corresponding blocks [1] [2] [3] [4] while in reality, bus pins are on the block boundaries and it is thus less accurate to connect buses to the block center in estimating bus length. Besides, we use general bus shapes to allow more flexible bus shapes and more flexible bus bending positions. We also improve the performance to give a solution with shorter total bus length.

In this thesis, we solve the fixed-outline bus-driven floorplanning problem with an objective to minimize the total bus length and interconnect cost. In the first approach, we focus on the bus length minimization problem. Instead of adding up all the bus length in the cost function of the SA process, we handle the bus length minimization problem globally in a more sophisticated way by using a bus-driven partitioning step. In this step, min-cut partitioning is performed recursively to reduce bus length. At the same time, we use a more accurate estimation algorithm

that connects buses to the block boundaries to estimate bus length. We can achieve shorter bus length on average by using this approach. Experimental results show that we can improve over the most closely related previous work [4] in terms of bus length, running time and dead space. Besides, we also improve over a most recent work [5] in terms of bus length on average.

In the second approach, we focus on the bus routability problem. In all the previous works [1] [2] [3] [4] [5], for the solutions with infeasible buses in some iterations of the SA process, a very simple method is used to handle it, that is, adding penalty to the cost function. Different from them, we handle the bus routability problem in a more sophisticated manner to increase the searching space and give more flexibility by changing the bus shape and the positions of the bus bendings. We present a floorplanner that can give a fixed-outline floorplanning solution including bus route that minimizes bus length, number of extra vias and interconnect cost. Our goal is to route all the bus components successfully in each iteration of the SA process of this floorplanner. L-shape bus component is used to route the buses with conflicts with other bus components. Experimental results show that we also improve over the most recent work [5] in terms of bus length and successful rate on average. Besides, we can improve over our first floorplanner in Chapter 3 and the modified floorplanner of [4] in terms of both bus length, running time and successful rate.

1.4 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 bus-driven floorplanning will be given in Chapter 2. In Chapter 3, we propose a method to solve the fixed-outline bus-driven floorplanning problem. After that, a method to solve fixed-outline bus-driven floorplanning with L-shape bus will be presented in

Chapter 4. Finally, a conclusion will be presented in Chapter 5.

□ **End of chapter.**

Chapter 2

Literature Review on BDF

Bus-driven floorplanning (BDF) considers bus assignment during floorplanning. In the BDF problem, we are given the following:

1. A set of blocks and each block has an area and an aspect ratio bound;
2. A set of buses and each bus has a width and a bus net;

We need to decide the position of each block and layout the buses on some metal layers. No overlapping between any two blocks and between any two buses on the same layer are allowed. The objective is to minimize chip area and the total bus area.

Eight terminologies related to BDF are given in Table 2.

2.1 Zero-Bend BDF

2.1.1 BDF Using the Sequence-Pair Representation

The work [1] is the first piece of work on BDF. In this paper, the block information and bus information are given. The goal is to decide the position of each block

Table 2.1: Terminologies for BDF

Terminology	Definition
Blocks	A whole circuit is partitioned into several sub-circuits in the partitioning step of the Physical Design cycle. Each sub-circuit is called a block.
Buses	A bus is a collection of wires, which can be used to carry signals among different blocks. Buses are of different widths.
Bus Pins	Pins for connect buses to blocks.
I/O pins	Pins for chip's input and output.
Nets	Each net connects several different blocks and I/O pins.
Netlist	A list of nets.
Vias	A via is a small opening in an insulating oxide layer that allows a conductive connection between different layers.
Routability	Routability refers to the possibility of completing all the connections between blocks pins.

and the position of each bus. No overlapping between blocks and no overlapping between buses on the same layer are allowed. The objective is to minimize the chip area and the total bus area. The authors in this paper assume that there are two metal layers for bus routing, one horizontal and one vertical.

The approach is based on SA and the sequence-pair floorplan representation. They derived feasibility conditions on sequence pairs that give feasible BDF solutions by analyzing the relationship between bus ordering and block ordering in a floorplan.

The necessary condition on block ordering of feasible horizontal and vertical bus is given as follows. If the blocks of a bus violate block ordering in the sequence pair, the bus cannot be assigned.

Block Ordering Given a sequence pair (X,Y) and a bus $u = b_1, ..., b_k$, if u is feasible, then the ordering of the k blocks should be either the same or reverse in the two sequences X and Y. Furthermore if the k blocks appear in the same order in both X and Y, the orientation of u is horizontal; otherwise u is vertical.

If there is more than one bus, bus ordering need to be determined to route buses one by one and remove infeasible buses if needed. The bus ordering is influenced by the sequence pair. Four cases for bus ordering between two buses are derived. Suppose that the buses are horizontal, then there are totally four cases as illustrated in Figure 2.1. In case 1, bus u is above bus v and in case 2, bus u is below bus v. In case 3, the two buses cannot be routed at the same time. In case 4, the two buses simply do not have any ordering between them. p and q are subsequence pair for the blocks related to bus v and v.

As shown in case 3 in Figure 2.1, some buses cannot be routed successfully at the same time. A bus ordering constraint graph is constructed to detect infeasible buses. In this graph, each vertex represents a bus. If a block on bus u is above a block on bus v, one edge is added from u to v in the graph. After constructing the graph, cycle detecting is done on the graph. If any cycle exists, it means that there is at least one infeasible bus. A node deleting algorithm is proposed to remove infeasible buses. In this algorithm, a node with maximum degree is deleted. After infeasible buses are removed and a bus ordering for the feasible buses is obtained, the coordinates of blocks and buses are calculated. In the cost function, the chip area, total bus area and the number of infeasible buses are considered.

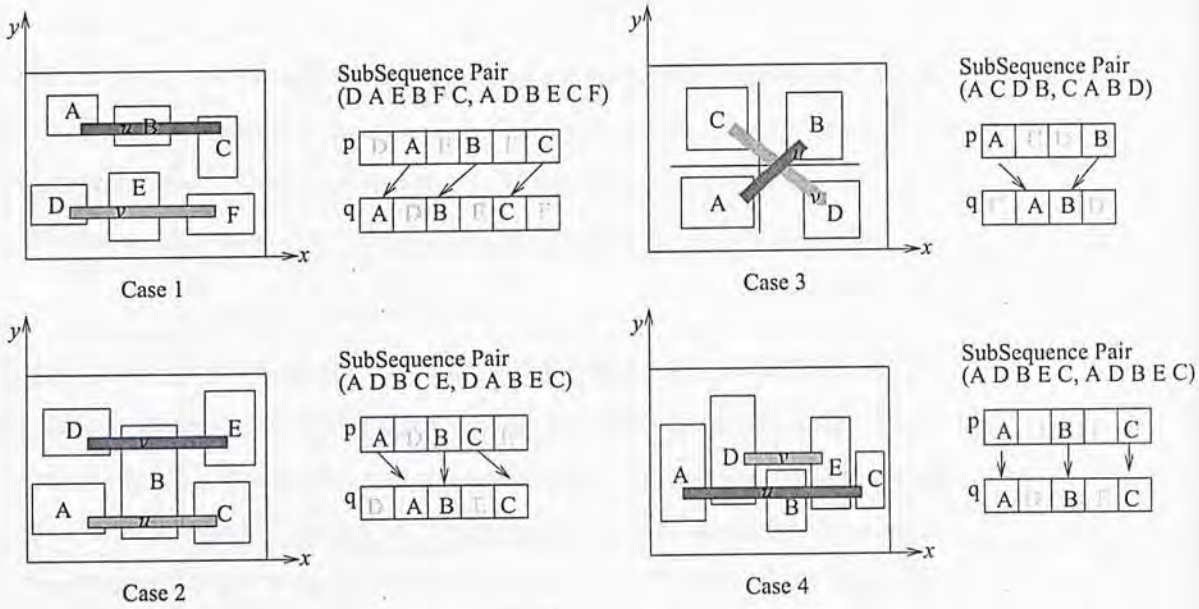

Figure 2.1: Cases of relative positions of two horizontal buses.

2.1.2 Using B*-Tree and Fast SA

In [2], the problem formulation is the same as that in [1]. The feasibility conditions of the B*-tree representation with bus constraints are explored and a BDF algorithm is developed based on the conditions and fast SA (Fast-SA). Three B*-tree properties for bus routing constraints are explored.

Property 1 In a B*-tree, the nodes in a left-skewed subtree may satisfy a horizontal bus constraint, i.e., the blocks can be aligned to allow a horizontal bus to pass through those blocks.

For a B*-tree, the left child n_j of a node n_i represents the lowest adjacent block b_j, on the right side of block b_i. Therefore, the blocks in a left-skewed subtree have horizontal relationships.

Property 2 By inserting dummy blocks of appropriate heights, it is guaranteed

that the nodes in a left-skewed subtree can satisfy the horizontal bus constraint.

Blocks are compacted to the bottom and left after packing. Therefore, the blocks associated with a left-skewed subtree of a B*-tree may not be aligned horizontally if the y-coordinates of some blocks are too small. To align those blocks, dummy blocks are inserted below them. The heights of the dummy blocks can be adjusted to satisfy the horizontal bus constraint.

Property 3 In a B*-tree, the nodes in a right-skewed subtree can guarantee the feasibility of a vertical bus.

For a B*-tree, the right child n_k of a node n_i represents the closest upper block b_k which has the same x-coordinate as block b_i. Therefore, the blocks in a right-skewed subtree are aligned with their left boundary. It is concluded that the nodes in a right-skewed subtree can guarantee the feasibility of a vertical bus if we assume that the minimum width of all the blocks on the bus net is larger than the width of the bus.

A B*-tree with a twisted-bus structure will be discarded during solution perturbation in the Fast-SA. To avoid bus overlapping, dummy blocks are inserted. B*-tree is first initialized as a complete binary tree at the beginning of the Fast-SA. In each iteration of the Fast-SA, after perturbation and non-dummy block packing, the bus structure in the B*-tree is checked. If there exists a twisted-bus structure in the B*-tree, the B*-tree is discarded and perturbed again. Otherwise, dummy blocks are inserted to appropriate nodes and the heights of the dummy blocks are adjusted to satisfy the horizontal bus constraints and to avoid bus overlapping. Then, the B*-tree with dummy blocks is packed again. Finally, the location of each bus is decided. In the cost function, the chip area, total bus area and number of infeasible buses are considered.

In addition, the method is extended to multi-bend bus solution. Firstly, the blocks on a bus is divided into several groups. Each group of the blocks forms a segment of the bus. After deciding the location of each segment, the length of each segment is simply extended to connect all segments to form a bus. The grouping of the blocks and the number of bendings are decided by the perturbation steps in Fast-SA. In this scenario, besides the chip area, the total bus area and the number of infeasible buses, the number of infeasible segments and the number of bus bendings are considered in the cost function.

2.2 Two-Bend BDF

In [3], the problem formulation is not exactly the same as that in [1]. Different from other previous work, the authors in the paper allow 0-bend bus, 1-bend bus and 2-bend bus. This method is based on a SA framework and the sequence-pair floorplan representation is used. The necessary conditions for feasible buses are derived, for which only 0-bend, 1-bend, and 2-bend are allowed. The details are given as follows.

0-Bend Bus Checking A 0-bend bus is actually a horizontal bus or a vertical bus. Given a sequence pair and a bus u_i that needs to go through blocks in $B_i = \{b_1, b_2, ..., b_k\}$, those blocks in B_i are extracted from the sequence pair to form two extracted sequences without altering their relative positions. sp_i is used to represent the subsequences extracted from the sequence pair for bus u_i. 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). For example, given a sequence pair $(DEFABC, ABCDEF)$ and two buses $u_0 = A, B, C$ and $u_1 = C, F$, the extracted sp_0 is (ABC, ABC) and sp_1 is (FC, CF). Therefore, u_0 can be realized using a 0-bend horizontal bus while u_1 can be realized using a 0-bend vertical bus.

This example is illustrated in Figure 2.2.

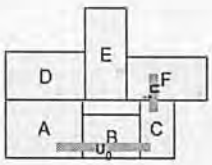

Figure 2.2: Two valid 0-bend buses, $\{A,B,C\}$ and $\{C,F\}$.

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 those in LCS) in the two sequences are in reversed order (vertical component).

This is possible because if taking the LCS as the horizontal component fails to form a valid L-shaped bus, taking any other shorter subsequences will also fail. If there exist more than one LCS, it has been shown that picking either one of them will work.

Note that even if a bus is consisted of one vertical component and one horizontal component only, there are still several possibilities. For example, the blocks may be in T-shape or +-shape, however, they are not considered in the approach of [3] . The authors claim that for some cases, a T-shaped bus can be changed into a valid 2-bend bus as shown in Figure 2.3. T-shaped buses that cannot be changed into a valid 2-bend bus are treated as invalid buses in [3].

2-Bend Bus Checking If the bus is found to be neither 0-bend nor 1-bend, it will be checked for 2-bend. There are several kinds of 2-bend buses, Z-shape (as shown in the upper right part of Figure 2.4), mirrored Z-shape (as shown in

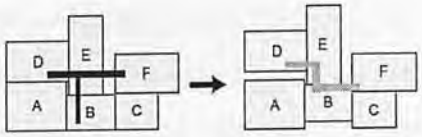

Figure 2.3: An example showing a T-shaped bus being able to change into a valid 2-bend bus.

the lower right part of Figure 2.4), C-shape (as shown in the lower left part of Figure 2.4), or mirrored C-shape (as shown in the upper left part of Figure 2.4). There will be two horizontal (vertical) components and one vertical (horizontal) component in the bus, denoted by HVH and VHV.

For the case of HVH, the vertical component of the bus will be first identified. Let the extracted sequence pair sp_i of bus u_i be (α, β), where α and β are strings of the block names. The vertical component can be found by finding the LCS in (α, β^R), where β^R denotes the reverse of the string β. The first block and the last block of the LCS will be kept for horizontal component checking.

After identifying the vertical component, the remaining blocks of the bus will be classified into different relationships with the vertical component. The relationships can be deduced from the sequence pair. There are totally eight relationships: 1) Upper, 2) UpperLeft, 3) Left, 4) LowerLeft, 5) Lower, 6) LowerRight, 7)Right and 8)UpperRight. In order to form a valid shape, no blocks should be identified in certain relationships. For example, to form a mirrored Z-shape, there should be no block on the upper-left and lower-right of the vertical component. Details are shown in Figure 2.4.

In each iteration of the SA process, the shape of a bus is first deduced by looking at the sequence pair of the floorplan. Whether there are buses that cannot be routed is then checked. The buses with conflicting constraints with other buses will be removed. A bus ordering between valid buses will be determined. The final

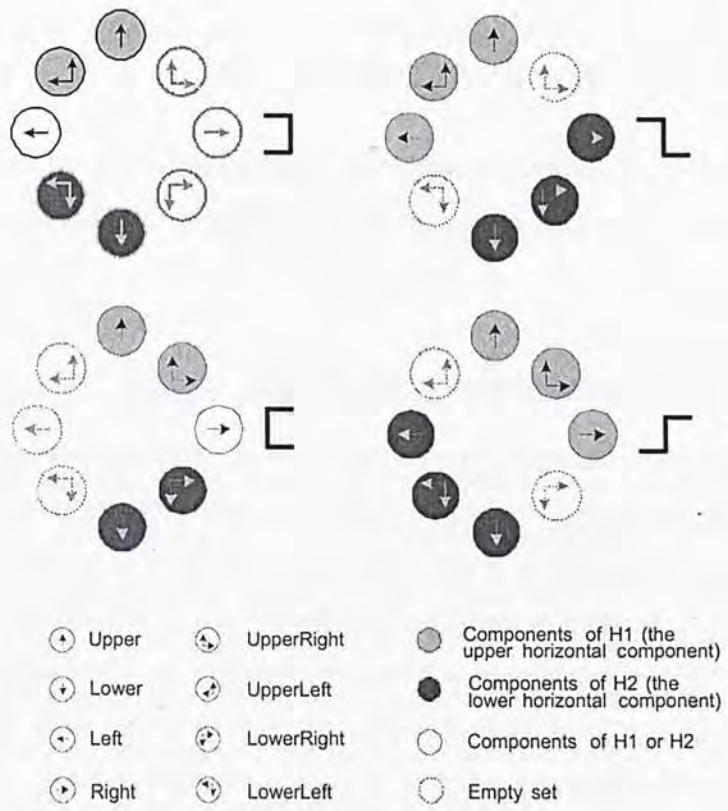

Figure 2.4: The necessary conditions of the position sets to form a valid 2-bend shape.

step is to realize the floorplan, i.e., obtaining the coordinates of the blocks and the buses.

2.3 TCG-Based Multi-Bend BDF

In [4], Transitive Closure Graph (TCG) representation is used. The problem formulation is similar to that in [1] except that the authors in the paper address the BDF problem under the constraint that all bendings must occur at the blocks on the bus net. There is no limitation on the bus shape and the number of bendings

as long as the above requirement on the bending positions is satisfied. A method to deal with placement constraints for bus routing in the TCG representation is proposed.

2.3.1 Placement Constraints for Bus

For each 0-bend horizontal (vertical) bus, they add a dummy block and some edges between related blocks into the vertical (horizontal) constraint graph.

Take a horizontal bus B with width t and a bus net N as an example. In order to allow the bus to pass through all its blocks in the final floorplan, they need to maintain a relative relationship between the modules in the vertical direction, i.e., the vertical overlap of the modules has to be at least the bus width t. This can be done by adding constraint edges to the vertical constraint graph G_v.

A dummy module m_d of height t and zero width to represent bus B is first added to G_v. Some constraint edges between m_d and each m_i in N are then added to G_v. In this case, the distance of m_i's lower right corner relative to m_d's lower left corner must be in the range of $[-h_i + t, 0]$, so a pair of constraint edges are added G_v:

1. An edge from m_d to m_i with weight $t - h_i$
2. An edge from m_i to m_d with weight 0

Similarly, for a vertical bus, a dummy module m_d of zero height and width t will be added to G_h. A pair of constraint edges will then be added between each m_i in N and m_d as follows.

1. An edge from m_d to m_i with weight $t - w_i$
2. An edge from m_i to m_d with weight 0

Figure 2.5 shows the vertical constraint graph G_v after inserting the constraint edges.

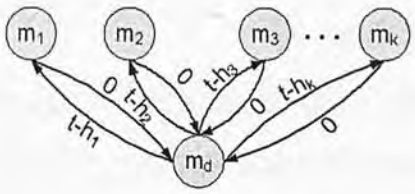

Figure 2.5: Constraint edges added to G_v for a horizontal 0-bend bus.

A multi-bend bus is formed by one or more 0-bend bus components. After decomposing a multi-bend bus into a set of 0-bend bus components, the corresponding sets of additional constraint edges for each component can be inserted into the constraint graphs as discussed in the previous paragraphs to align the blocks for the bus to pass through. Figure 2.6 shows a placement of four blocks and an L-shaped bus with two bus components, one horizontal and one vertical. The constraint graphs with the additional constraint edges are shown.

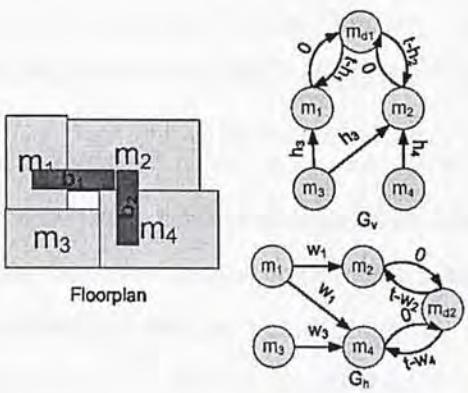

Figure 2.6: An example of handling multi-bend buses.

2.3.2 Bus Ordering

A proper bus order is needed to avoid bus overlapping. For some buses, their ordering can be deduced from the vertical constraint graph.

Given a floorplan of n modules $\{m_1, m_2 \ldots m_n\}$ with constraint graphs $G_h = (V, E_h)$ and $G_v = (V, E_v)$, the edges in E_h and E_v, representing the relative positions between the modules, may give a natural ordering between two buses B_1 and B_2 with bus nets N_1 and N_2 respectively as follows:

Case 1 B_1 is on top of B_2 when

1. B_1 and B_2 are both horizontal bus components, and
2. $\exists e_{ij} \in E_v$ such that $m_j \in N_1$ and $m_i \in N_2$.

Case 2 B_1 is on the left side of B_2 when

1. B_1 and B_2 are both vertical bus components, and
2. $\exists e_{ij} \in E_h$ such that $m_i \in N_1$ and $m_j \in N_2$.

Figure 2.7 shows an example in which a natural ordering can be deduced from the vertical constraint graph. In this example, the bus going through block m_1 and m_2 must be above that going through m_4 and m_5 since m_4 is required to be placed below m_2.

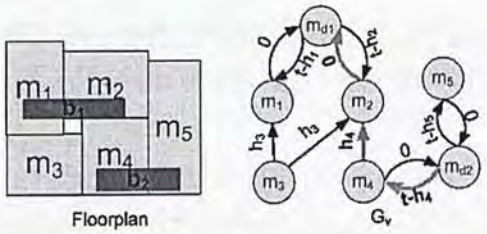

Figure 2.7: Two horizontal buses with a natural ordering deduced from the constraint edges.

For those bus pairs which do not have such natural orderings, their orderings is assigned explicitly if they may overlap. There are only two cases that two bus components b_1 and b_2 may overlap:

Case 1 $N_1 \cap N_2 \neq \emptyset$, i.e., b_1 and b_2 share at least one module.

Case 2 $N_1 \cap N_2 = \emptyset$ and $\exists m_i \in N_1$ and $m_j, m_k \in N_2$ or $\exists m_i \in N_2$ and $m_j, m_k \in N_1$ such that e_{ji} and $e_{ik} \in E_h$ (or e_{ji} and $e_{ik} \in E_v$), i.e., the modules of b_1 and b_2 interleave with each other in the x-direction (or y-direction).

In these two cases, an explicit bus ordering is imposed to prevent overlapping. Suppose t_1 and t_2 are the widths of b_1 and b_2 respectively and m_{d1} and m_{d2} are their corresponding dummy modules in the constraint graphs. An explicit bus ordering can be enforced as follows:

1. When b_1 and b_2 are both horizontal, an edge from m_{d1} to m_{d2} with weight t_1 is added or an edge from m_{d2} to m_{d1} with weight t_2 is added to G_v
2. When b_1 and b_2 are both vertical, an edge from m_{d1} to m_{d2} with weight t_1 is added or an edge from m_{d2} to m_{d1} with weight t_2 is added to G_h

Figure 2.8 shows an example of how bus overlapping can be prevented by imposing an explicit bus ordering. In this example, the overlapping between the two horizontal bus components is removed by adding an edge of weight t_2 from dummy node m_{d2} to node m_{d1} in G_v.

The method is based on a SA framework. In each iteration of the SA process, a pair of reduced constraint graphs are constructed. For each bus, a set of bus components is obtained by applying a modified minimum spanning tree algorithm on the common graph that is constructed from the pair of reduced constraint graphs. Then, for each bus component, a dummy block and a set of constraint

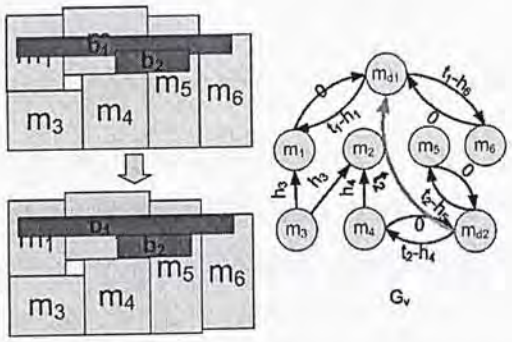

Figure 2.8: Prevention of bus overlap by imposing explicit bus ordering. In this example, b_1 is connecting m_1 and m_6, and b_2 is connecting m_4 and m_5.

edges between dummy block and blocks in the bus component are added to the corresponding constraint graph to make sure that the bus component can pass through the blocks. After that, the bus feasibility is checked. If the bus is infeasible, the dummy block and corresponding edges will be deleted from the graph. After adding placement constraints for all the buses, a set of constraint edges between different dummy blocks are added to the corresponding graph to impose explicit bus ordering for those bus pairs which do not have natural orderings. Finally, a single source longest path algorithm is performed to determine the positions of the blocks and the buses. The cost function considers the chip area, the total bus area and the number of infeasible buses.

At the end, a soft module adjustment process is done to minimize the total chip area. It is another SA process. In each iteration, a block on the critical path is chosen and its width or height is changed a little bit.

2.4 Bus-Pin-Aware BDF

In [5], the problem formulation is a little bit different compared with other previous works. Besides block information and bus information, the position of each bus

pin is also given. The objective is to decide the position and orientation of the bus pins on each block and determine the routing path of each bus such that there is no overlapping between different bus on the same layer. The floorplanner needs to minimize the chip area and the total bus area. In the paper, it is assumed that the two metal layers for bus routing are unreserved layers.

Wu *et al.* [5] considered the impacts of bus pins in their BDF algorithm. Bus bendings are not restricted to occur at only the blocks in the bus net. Diagonal connection between different blocks is considered to make the bus shape more flexible. The number of vias is reduced by assigning the two components of a diagonal bus to the same layer. An algorithm is proposed to minimize the bus length deviation for the signal integrity issues. In addition, two bus length reduction algorithms to improve the total bus length are presented. In the first algorithm, the overlapping between different buses is eliminated when constructing minimum spanning trees. In the second algorithm, each horizontal (vertical) bus is moved towards the direction with the maximum number of vertical (horizontal) buses intersecting with it.

The method is based on a SA framework and the sequence pair representation is used. In each iteration of the SA, a modified Prim's algorithm is used to derive the bus routing topologies. In the modified Prim's algorithm, diagonal connection between different blocks and the capacity of each block are considered. The weight of each edge is derived from the distance between two blocks. The bus length is minimized by reducing the redundant parts in the bus routing topologies. The second step in each iteration is floorplan realization. This step is further broken down to three sub-steps, including bus ordering and coordinates determination, bus length reduction, and layer assignment.

In the first sub-step, an ordering constraint graph (OCG) is first constructed to determine the bus ordering. In OCG, each bus is represented by a vertex. The

ordering of any two buses is derived from the relative position of those blocks on the buses. If there are vertical constraints between those blocks, there will be vertical constraint between the corresponding buses and there will be an edge between the two buses. The bus ordering is determined by deleting vertices and their edges from the OCG iteratively. The vertices with zero out-degree and their corresponding edges are first removed from the OCG. However, some buses may conflict with other buses. In this scenario, there is no vertex with zero out-degree. One of the vertices with minimum out-degree is regarded as an infeasible bus and this vertex and its corresponding edges are deleted from the OCG. This process is repeated until all the vertices are deleted from the OCG. The ordering of the feasible buses is derived from the order in which they are removed from the OCG. After determining the bus ordering, the coordinates of each bus is calculated one by one according to the bus order. The coordinates of each horizontal (vertical) bus is $y_{max}(x_{max}) = max\{y_i(x_i)|i = 1,2,...,k\}$, where k is the number of blocks passed by the bus. $y_i(x_i)$ is the $y(x)$-coordinate of the block. They change the coordinates of blocks slightly if needed to make sure that the bus can pass through the blocks.

In the second sub-step, to reduce the bus length, they move each horizontal (vertical) bus toward the direction with the largest number of vertical (horizontal) buses intersecting with it. In the third sub-step, a graph is first constructed, where each vertex represents a bus and there is an edge between two vertices if the two corresponding buses have overlapping with each other. A graph coloring algorithm is used to assign each bus to one of two metal layers. A vertex with maximum degree is assigned to layer 1 and all its neighbors are assigned to layer 2. Then starting from one of its neighbors, all its neighbors are assigned to layer 1. The process is repeated until all vertices are assigned to one of the two layers. If there is an odd cycle in the graph, some buses cannot be assigned by using the above two coloring algorithm. In this scenario, one of the corresponding buses are regarded

as infeasible. Therefore, not all the buses can be routed successfully in some cases.

After the SA process, the orientation of each bus pin is determined and the bus length deviation is minimized based on the solution from the SA process. Bus length deviation of each bus $B_i = \{b_1, b_2, .., b_k\}$ is calculated as the sum of bus length deviations of each bus segment $s_j = \{b_j, b_{j+1}\}, j = 1, ..., k-1$ between two blocks. They summarize 24 general patterns for all possible bus shapes between any two blocks. For each general pattern, its bus length deviation can be calculated. The possible patterns can be obtained from the initial positions of the bus pins on the blocks. They start from s_1 and choose the pattern with minimum accumulated deviation at block b_2 and determine the orientation of bus pin on block b_1. The pattern with minimum accumulated deviation at block $b_j, j = 3, 4, ..., k$ is then chosen and the orientation of the bus pins on each block is determined.

At the end, a soft module adjustment process is done to minimize chip area. It is the same as the soft module adjustment in [4].

2.5 Summary

In this chapter, the previous works on BDF are reviewed. All of the previous works [1] [2] [3] [4] [5] focus on minimizing the chip area and total bus length. On the bus routability problem, it was assumed that only 0-bend buses are allowed [1] [2]. Later, other researchers [3] assumed only 0-bend buses, 1-bend buses and 2-bend buses. All these works explored the conditions for feasible bus routing on the floorplan representations they used, such as sequence pair and B*-tree representation. The authors of [4] explored methods that allow no limitation neither on bus shape nor on the number of bus bendings as long as the bendings occur at blocks on the corresponding buses. In [5], besides allowing no limitation neither on bus shape nor on the number of bus bendings, the bus bendings are not restricted to occur at blocks on the corresponding buses. On the bus length

minimization problem, no special method was explored to minimize bus length in the beginning. Later, some researchers [4] [5] explored the modified minimum spanning tree construction algorithms to minimize bus length.

□ End of chapter.

Chapter 3

Fixed-Outline BDF

3.1 Introduction

Bus routing has become an important research topic as the interconnection of chips become increasingly congested. It would be favourable to consider bus routing in a floorplanner that minimizes chip area, interconnect cost and bus length. BDF considers bus assignment during floorplanning.

The aim of our floorplanner is to give a fixed-outline floorplan solution including bus routes that minimizes bus length and interconnect cost. We propose two steps to minimize bus length. Firstly, recursive bi-partitioning is used at the beginning to handle the problem globally. We minimize the bus length by performing a min-cut partitioning. Secondly, we devise a modified minimum spanning tree (MST) algorithm to generate the topology of a bus to reduce the bus length. Experimental results show that we can improve over the most closely related previous work [4] in terms of bus length, running time and deadspace. Besides, we also improve over another recent work [5] in terms of bus length on average.

3.2 Problem Formulation

In this fixed-outline BDF problem, we are given the following:

1. A set of blocks and each block has an area and an aspect ratio bound
2. A set of buses and each bus has a width and a bus net
3. A set of nets
4. A set of pins and each pin has coordinates
5. Fixed outline $W \times H$

The aim is to obtain a fixed-outline floorplan solution including bus routes that minimizes the bus length and interconnect cost. We need to decide the position of each block and layout the buses on two metal layers, one horizontal and one vertical. No overlapping between any two blocks and between any two buses on the same layer are allowed. Besides, in order to minimize the number of vias in the chip, all bendings of the buses can only occur on the blocks on the corresponding bus nets [4]. One example is shown in Figure 3.1 to illustrate two possible BDF solutions. In this example, there are four blocks b_1, b_2, b_3, b_4 and one bus B_1 connecting b_1 and b_4. The first BDF solution is shown in Figure 3.1(A). B_1 has a bending on b_3 which does not belong to B_1. Therefore it is not a feasible solution according to the problem definition. The second one is shown in Figure 3.1(B). B_1 is a horizontal bus and it is a feasible solution.

3.3 The Overview of Our Approach

An overview of our approach is given in Figure 3.2. Our approach is composed of four steps. The first step is partitioning. Blocks are partitioned recursively into

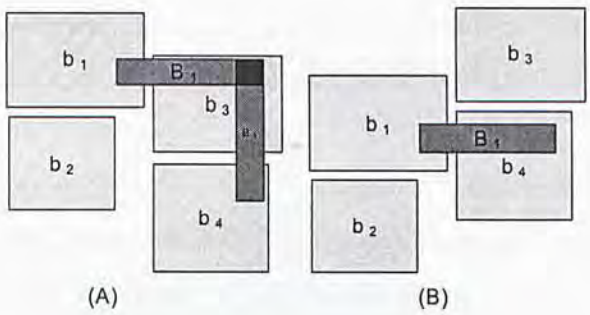

Figure 3.1: Examples of possible BDF solutions

several groups and the result of this partitioning step is used as the initial candidate solution of the next floorplanning step. The second step is a SA process. The objective of this step is to find a floorplan solution with feasible bus routing, minimizing the bus length, chip area and wire length. In the third step, another annealing process is used to handle the fixed-outline constraint based on the solution obtained in the second step. Finally, we layout the buses and decide the positions of the bus pins in the last step. The details are discussed in Section 3.4, Section 3.5, Section 3.6 and Section 3.7.

3.4 Partitioning

Both the block positions and bus topology affect the bus length. To minimize bus length, it will be favourable that all the blocks in the same bus net are close to each other. However, a block can be in several different bus nets. We use partitioning to handle this global bus connection problem. A bus-driven partitioning will first be performed to partition blocks into several groups according to the bus net information. The result of this partitioning step is used as the initial candidate solution of the next floorplanning step.

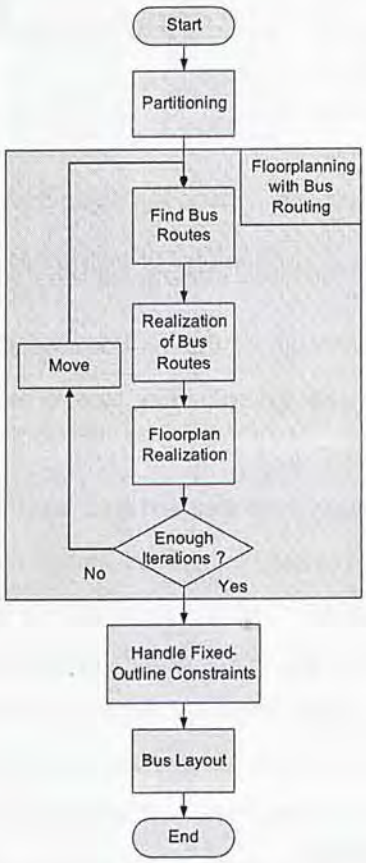

Figure 3.2: Overview of our approach

3.4.1 The Overview of Partitioning

In the partitioning step, blocks are partitioned level by level. In the first level, blocks are partitioned into two groups. One is on the left (lower) side and the other is on the right (upper) side. In the next level, each of the two groups are further partitioned into two groups. One is on the lower (left) side and the other is on the upper (right) side. The process is repeated until the number of blocks in each final group is less than a certain value.

To partition a set of blocks into two groups, a hypergraph G is first built according to the *Exact Net-Weight Modeling* [8] that maps bus length to cut cost of the

corresponding hypergraph. This is to guarantee that finding a min-cut of the hypergraph is equivalent to minimizing the total bus length estimated using the half parameter wire length (HPWL). The details of building this type of hypergraph will be given in Section 3.4.2. The method *hMetis* [7] is then used to partition the hypergraph. The final result of this partitioning step is used as the initial candidate solution of the annealing process in the floorplanning step. [1]

3.4.2 Building a Hypergraph G

To partition a set of blocks into two groups, a hypergraph G is first built according to the *Exact Net-Weight Modeling* [8] that maps bus length to cut cost of the corresponding hypergraph. Suppose that we want to partition a group of blocks $\{b_1, b_2, ..., b_n\}$ by a vertical cut. The group of n blocks are packed in a region with a fixed width and height. Let **A** denote the region. Suppose that there is a bus $B = \{b'_1, b'_2, ..., b'_q\}$ that has at least one of its q blocks in the set $\{b_1, b_2, ..., b_n\}$. Let $B_1 = \{b'_1, b'_2, ..., b'_j\}$ denote the set of j blocks in the set $\{b_1, b_2, ..., b_n\}$ and $B_2 = B - B_1 = \{b'_{j+1}, .., b'_q\}$. The construction of the corresponding hypergraph G is given as follows.

Vertices of the Hypergraph G

Each block in the set $\{b_1, b_2, ..., b_n\}$ corresponds to a vertex in **G** and the weight of a vertex is the area of the corresponding block. Two dummy vertices are added to **G** and the weight of each dummy vertex is zero. One dummy vertex b_l is fixed

[1] Our floorplanner is different from many other partitioning-based floorplanners as follows. By the end of the partitioning step, we get several groups of blocks. We will then merge all the groups back to get an initial candidate solution for the subsequent BDF process, instead of treating each group separately. It is important for us to treat all the partitions simultaneously since we are doing BDF and need to make sure that all the buses are routable at the end while the buses may cross between several partitions.

on the left (upper) part of the region and the other dummy vertex b_r is fixed on the right (lower) part of the region. Their positions are at the center of the left sub-region c_l and the center of the right sub-region c_r respectively. b_l and b_r are used to discriminate the left sub-group and the right sub-group of **A**. After partitioning (using *hMetis*), all the blocks that are in the same group as b_l belong to the left (upper) part of the region and all the blocks that are in the same group as b_r belong to the right (lower) part of the region.

Hyperedges of the Hypergraph G

According to the *Exact Net-Weight Modeling*, each bus net will be represented by two hyperedges E_1 and E_2 in **G**. Assume that all the blocks in a region are at the center of the region. Each region is divided evenly because we want to keep two sub-groups balanced in terms of area.

An example is given to show how to construct E_1 and E_2 in Figure 3.3. In the example, we assume that $j = 2, q = 4$, that is, $B = \{1,2,3,4\}$, $B_1 = \{1,2\}$ and $B_2 = \{3,4\}$.

The first hyperedge E_1 corresponding to the bus net B is composed of the blocks in B_1 and b_l (or b_r). Suppose that W_l is the HPWL value for bus B when we assume that all the blocks in B_1 are at c_l (the center of the left part of region **A**) as shown in Figure 3.3(a). Suppose that W_r is the HPWL value for bus B when we assume that all the blocks in B_1 are at c_r (the center of the right part of region **A**) as shown in Figure 3.3(b). If $W_l <= W_r$, hyperedge E_1 is composed by the blocks in B_1 and b_l. Otherwise, hyperedge E_1 is composed by the blocks in B_1 and b_r. The weight of $E_1 = max\{W_l, W_r\} - min\{W_l, W_r\}$. For the example in Figure 3.3, $W_l < W_r$, the first hyperedge $E_1 = \{1,2,b_l\}$. The weight of E_1 is defined as $W_r - W_l$.

The second hyperedge E_2 corresponding to B is composed of the blocks in B_1.

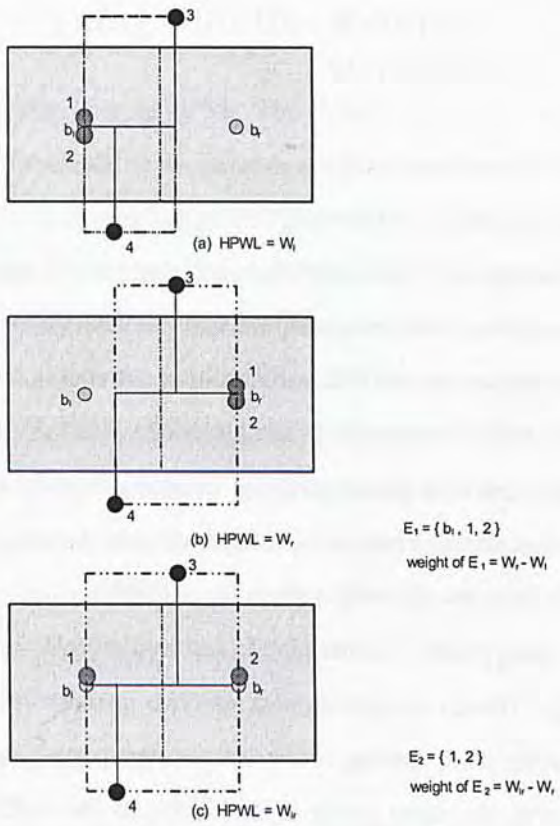

Figure 3.3: An example of constructing E_1 and E_2

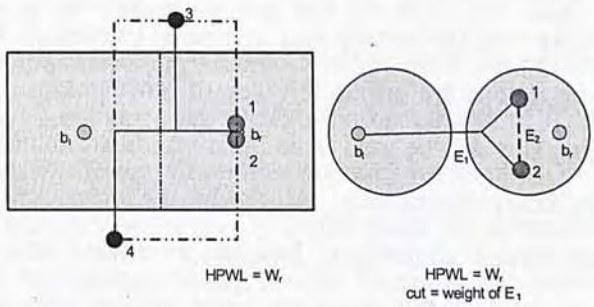

Figure 3.4: An example of cut

Suppose that W_{lr} is the HPWL value for bus B when we assume that some blocks in B_1 are at the position c_l while the others are at c_r as shown in Figure 3.3(c).

The weight of E_2 is defined as $W_{lr} - \max\{W_r, W_l\}$. For the example in Figure 3.3, $E_2 = \{1,2\}$ and $W_l < W_r$, so the weight of E_2 is $W_{lr} - W_r$.

By constructing two hyperedges for each bus net, we can map bus length to cut cost. For each bus , we can guarantee that $HPWL = min\{W_l, W_r\} + cut$. Take figure 3.4 as an example. HPWL is W_r and the cut cost is the weight of E_1, that is, $W_r - W_l$. So we can obtain the below equations. Since W_l and W_r are constants for a given partition, it is guaranteed that finding a min-cut of the hypergraph is equivalent to minimizing the total bus length estimated using HPWL.

$min\{W_l, W_r\} + cut = W_l + W_r - W_l = W_r = HPWL$.

After the hypergraph is constructed, the method *hMetis* is used to partition the hypergraph. Blocks are partitioned into two groups. All the blocks that are in the same group as b_l belong to the left (upper) part of the partition. All the blocks that are in the same group as b_r belong to the right (lower) part of the partition. The process is repeated until the number of blocks in each group is less than a certain value. In the end of the partitioning step, we obtain several groups of blocks with the vertical and horizontal constraints between them. For example, some groups are on the left side of other groups. The final result of this partitioning step will be used as an initial candidate solution of the annealing process in the floorplanning step. All the blocks are packed according to their horizontal and vertical constraints. Besides, we assume that the blocks in each group are packed horizontally since the initial position of each block in the same group does not affect the bus length a lot. We then use a SA process starting with that initial solution of all the blocks in each group being packed horizontally to search for a better solution.

3.5 Floorplanning with Bus Routing

The floorplanning step is done by SA. The objective of this step is to find a floorplan solution with feasible bus routing and minimizing the bus length, chip area and wire length. In each iteration of this process, the topology of each bus is generated by the method to be described in Section 3.5.1. According to the bus topology generated, we obtain several bus components. The bus routing constraints (i.e. aligning blocks to allow buses to pass through them and assigning bus ordering to avoid bus overlapping) are handled by adding a dummy block and some constraint edges into the horizontal constraint graph G_h or the vertical constraint graph G_v for each bus component. Details will be presented in Section 3.5.2. Finally, the coordinates of the blocks and the bus components are calculated by performing a single source longest path algorithm in G_h and G_v.

3.5.1 Find Bus Routes

Both the block positions and bus topology affect the bus length. A new algorithm is proposed to generate a good topology for a bus net. For each candidate floorplan solution, we are given a sequence pair and the width and height of each block. The objective is to connect all the blocks in the same bus net and minimize the total bus length. Buses are connected to the boundaries of their corresponding blocks. Our work is different from most previous works on BDF in the following aspect. In most previous works [1] [3] [2] [4], buses are assumed to be connected to the centers of their corresponding blocks. In reality, bus pins are on the block boundaries. It is thus more accurate to connect buses to the block boundaries in estimating the bus length.

We generate a bus topology by constructing a minimum spanning tree (MST) T in the combined constraint graph $G_c = G_h \cup G_v$. There are two issues that we

need to take special care of. Firstly, since buses are connected to the boundaries of their corresponding blocks, it is not accurate to use the distance between the lower left corners of two blocks as the weight of the corresponding edge in G_c. Secondly, some MST do not correspond to any feasible bus because we do not allow too many bus components of the same bus net passing through a block as we want to minimize the number of vias used. The number of bus components of the same bus net passing through a block should not exceed a maximum number, called the capacity of the block.

A modified MST algorithm is used to find the topology of a bus. The modified MST algorithm is similar to the Prim's algorithm except that the weight of each edge is updated dynamically. In this modified MST algorithm, the weight of an edge e_{ij} between block b_i and block b_j is the distance between a selected point on one block and a boundary of the other block. Block b_i's selected point is decided when b_i is added into the MST and the weight of each edge is calculated dynamically when constructing the MST. The selected point of the first block is its upper right corner. If block b_i is not the first block, its selected point is the point that is on b_i's boundary and nearest from b_j's selected point.(Suppose the edge e_{ij} from b_i to b_j is the edge with minimum weight.) We will also consider the capacity of each block. In our method, we assume that the capacity of each block is one. That means, for each bus net, each block can be passed through by at most one horizontal bus component on its left side, one horizontal bus component on its right side, one vertical bus component on its upper side and one vertical bus component on its lower side.

Similar to the Prim's algorithm, a heap data structure H is used to store elements $(b_j, e_{ij}, weight\ of\ e_{ij})$ for an edge e_{ij} connecting b_i and b_j. The modified MST algorithm has four steps. Its details are shown in Algorithm 1. Firstly, the block that is nearest from the point $(0,0)$ is selected to be the starting block. Its

selected point will then be decided. A selected point a_i of a block b_i can only be on b_i's boundary. For the starting block, its upper right corner will be chosen as its selected point. Now, the tree T contains only the starting block. Secondly, the weights of the edges connecting to the blocks in T are updated. The weight of an edge e_{ij} between block b_i and block b_j is the distance between the selected point on one block and a boundary of the other block, and is calculated as follows. If b_i has been added into T and b_j is not, the weight of e_{ij} is the nearest distance between b_i's selected point a_i and one of the four boundaries of block b_j. An element $(b_j, e_{ij}, weight\ of\ e_{ij})$ will then be inserted into heap H. Thirdly, the nearest block that satisfies the capacity constraint will be chosen to be added into T. The position of its selected point is then decided. Details of this step are explained as follows. We select the first element $(b_k, e_{hk}, weight\ of\ e_{hk})$ from heap H. If block b_k is not in T, the capacity of b_h and b_k will be checked. If the capacity limit is not violated, we can add b_k and edge e_{hk} into T. Otherwise, the process is repeated for the next min element in H until we choose a block to add it into T or when H is empty. If H is empty, no feasible bus topology is generated. After we choose a block, the block's selected point will be decided. The method of finding selected point is shown in an example in the next paragraph. The second step and the third step are repeated until all the blocks in the bus net are included in T. Note that this MST construction will fail sometime. Then the bus is considered as an infeasible bus.

An example is given in Figure 3.5. Suppose that the bus net is $B = \{b_1, b_2, b_3, b_4\}$. As shown in Figure 3.5(a), block b_1 is the nearest block from the point (0,0). Block b_1 is thus the starting block. Its selected point a_1 is its upper right corner. Now, block b_1 is in T and the other three blocks b_2, b_3, b_4 are not. We can see that the nearest distance between a_1 and those three blocks is d_{13}, d_{14} and d_{12} respectively. Since d_{13} is the smallest, block b_3 and edge e_{13} are selected to be added into T.

Algorithm 1 Modified MST Algorithm

1: select a starting block b_i and choose its upper right corner as its selected point
2: add the starting block b_i into T
3: **while** number of blocks in T is less than n **do**
4: /*n is the number of blocks in the bus*/
5: **for** each block b_j not in T **do**
6: calculate the weight of edge e_{ij} between b_i and b_j
7: insert $(b_j, e_{ij}, weight\ of\ e_{ij})$ into heap H
8: **end for**
9: **while** TRUE **do**
10: **if** H is not empty **then**
11: get the first element $(b_k, e_{hk}, weight\ of\ e_{hk})$ in H
12: **if** b_k is not in T **then**
13: **if** neither b_h's capacity limit nor b_k's capacity limit is violated /*e_{hk} is the edge between b_h and b_k*/ **then**
14: select b_k and e_{hk}
15: goto line 22
16: **end if**
17: **end if**
18: **else**
19: return False (bus topology generation failed)
20: **end if**
21: **end while**
22: add b_k and e_{hk} into T, decide b_k's selected point
23: $i = k$
24: **end while**

Block b_3's selected point is chosen to be at a_3 as shown in the Figure 3.5(a). After adding b_3 into T, the weights of e_{34} and e_{32} are calculated. We then insert (b_4, e_{34}, d_{34}) and (b_2, e_{32}, d_{32}) into the heap H. Four edges in H are shown in Fig-

ure 3.5(b). We can see that block b_4 is the nearest one and the capacity constraint is satisfied, so b_4 and the edge e_{34} are added into T. The position of its selected point is chosen to be at a_4. Similarly, block b_2 and edge e_{42} are added into T as shown in Figure 3.5(c). Finally, a MST is constructed and a good bus topology is generated as shown in Figure 3.5(d).

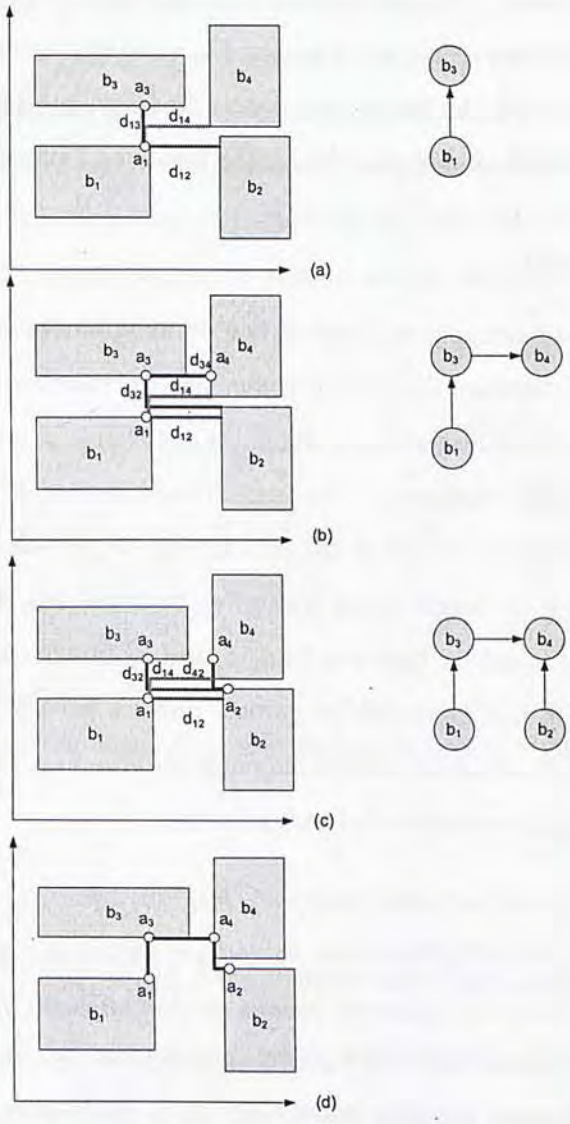

Figure 3.5: An example of the modified MST algorithm

3.5.2 Realization of Bus Routes

In this section, we use the basic techniques from [4] in bus routing to do the realization of bus routes. We consider how to align blocks to allow bus components to pass through. Bus ordering is considered to avoid bus overlapping on the same metal layer. Since the techniques are already described in [4] in a formal way, we will just describe them with examples in the following 2 sub-sections.

Block Alignment

We will give an example to illustrate how to align blocks in the same bus net component to guarantee that the bus component can pass through them. Suppose that we want to handle the block alignment constraint for a horizontal bus $B = \{b_1, b_2, b_3\}$ and its width is w. The vertical overlap of b_1, b_2 and b_3 must be at least the bus width w so that B can pass through b_1, b_2 and b_3. We first add a dummy block b' of height w and width 0 to represent bus B. Some constraint edges between b' and the three blocks b_1, b_2 and b_3 are then added to the vertical constraint graph G_v. Note that the vertical distance from b_i's lower left corner to b''s lower left corner must be in the range of $[0, h_i - w]$, $i = 1, 2, 3$. A pair of constraint edges are added to G_v to achieve this:

1. Edges from b' to b_i with weight $w - h_i$, $i = 1, 2, 3$.
2. Edges from b_i to b' with weight 0, $i = 1, 2, 3$.

Similarly, we can handle the block alignment constraint for a vertical bus by adding a dummy block and a set of constraint edges to G_h. Multi-bend bus can be handled by decomposing it into a number of horizontal and vertical bus components and the block alignment constraint for each bus component is handled by the above method.

For example, three bus components are generated in the example shown in Figure 3.5 after the finding bus routes step. Two of them $C_1 = \{b_1, b_3\}$ and $C_2 = \{b_2, b_4\}$ are vertical components and one of them $C_3 = \{b_3, b_4\}$ is a horizontal bus component. For each bus component, one dummy block and two pairs of edges will be inserted to allow each bus component to go through their blocks as shown in Figure 3.6. Take C_3 as an example. A dummy block b_3' is inserted to the vertical constraint graph. A pair of edges between b_3' and b_3 are inserted to the vertical constraint graph according to the above method. Besides, a pair of edges between b_3' and b_4 are inserted into the vertical constraint graph. In this way, b_3 and b_4 are aligned to allow bus component C_3 to pass through. Similarly, two dummy blocks and four pairs of constraint edges are inserted into the horizontal constraint graph for bus component C_1 and C_2.

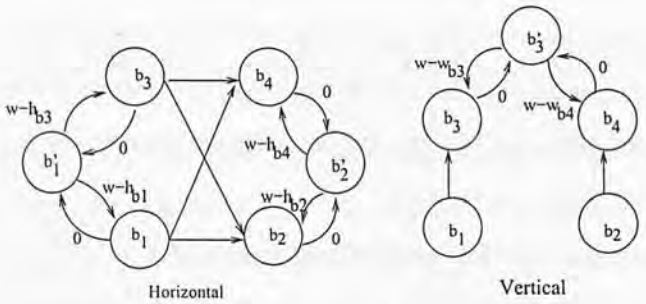

Figure 3.6: Block alignment for the example in Figure 3.5

Bus Ordering

A proper bus order is needed to avoid bus overlapping. We will give the following two examples to illustrate how to impose bus order. Consider two vertical bus components, $B_1 = \{b_1, b_2\}$ and $B_2 = \{b_3, b_4\}$. If there is an edge e_{13} from b_1 to b_3 in the horizontal constraint graph G_h, we can deduce a natural bus ordering, i.e., B_1 must be on the left of B_2. In this case, we do not need to impose an ordering

between B_1 and B_2 because we can deduce a bus ordering from the constraint obtained from the sequence pair. An example is shown in Figure 3.6. b_1 is on the left of b_2, thus bus component C_1 is on the left of C_2. For those bus pairs whose ordering cannot be deduced naturally from the constraint graphs, we need to assign bus orderings for them if they may overlap. Let's consider another example. In this example, suppose there are two vertical bus components $B_1 = \{b_1, b_2\}$ and $B_2 = \{b_2, b_4\}$ and both vertical bus components pass through b_2. We need to assign a bus ordering for B_1 and B_2 in order to avoid overlapping between B_1 and B_2 because their ordering cannot be deduced naturally from the constraint graphs. In this case, we will assign B_1 on the left of B_2 or B_2 on the left side of B_1 randomly and uniformly. We can then impose orderings for vertical bus components in G_h by adding edges between the dummy blocks of the corresponding bus components. For horizontal bus components, similar steps are taken to impose a bus ordering.

3.5.3 Details of the Annealing Process

In the SA process, the objective is to find a floorplan solution with feasible bus assignment and to minimize the bus length, total area and wire length.

Initial sequence pair

The initial sequence pair is obtained from the result of the partitioning step as follows. Each group in the final partitioning level is treated as a super block. The initial sequence pair representation for these super block packing can be obtained easily from the relationship of these super blocks in the partitioning stage. Within each super block, all the blocks are packed horizontally.

Random move

The following set of moves is used.

1. Rotate a block;
2. Interchange two blocks in the first sequence;
3. Interchange two blocks in both sequences.

In moves 2 and 3, besides choosing two blocks in the same partition, we can also choose two blocks from two partitions if none of them is on any bus net from different partitions. In this way, the interconnect cost can be reduced without changing the bus length significantly.

Cost function

In the annealing process, we want to minimize the total area, bus length and wire length with a feasible bus connection. The cost function is defined as:

$$Cost = \alpha Cost_{Area} + \beta Cost_{bus} + \gamma Cost_{wire} + \delta Cost_p \tag{3.1}$$

where $Cost_{Area}, Cost_{bus}, Cost_{wire}$ are the total area, total bus length and total wire length of the floorplan solution respectively. The total area is minimized because we want to search for a solution with feasible bus routes, minimum chip area, minimum bus length and minimum wire length in this step. The wire length is estimated with HPWL. $Cost_p$ is the penalty for solutions with infeasible bus routing and it is the number of infeasible buses. (A bus is regarded as infeasible if the bus cannot be routed.) The parameters α, β, γ and δ are weights obtained by performing random walk at the beginning of the annealing process. With this cost function, we minimize the total area of the packing first, and the fixed-outline constraints will be handled by a post-processing step as described below.

3.6 Handle Fixed-Outline Constraints

After a good floorplanning solution is obtained from the SA process in Section 3.5, the fixed-outline constraint will be handled on the solution in a post-processing step. We change the aspect ratios of some blocks and use another SA process to search for good block dimensions. In this post-processing step, the dimensions of the soft blocks are adjusted in the move operations of the annealing process. In each iteration, we choose a block lying on the critical path and change its aspect ratio slightly. The cost function $Cost_s$ is defined as in [9].

$$Cost_s = \max\{0, h - H\} + \max\{0, w - W\}, \tag{3.2}$$

where W and H denote the width and height of the given outline.

3.7 Bus Layout

In the floorplanning step, each bus net is decomposed into several horizontal bus components and vertical bus components. Figure 3.7 is one result of the SA process. We can see that some bus components are not connected to each other. Take bus $B_5 = \{4, 9, 12, 14\}$ as an example. It has three bus components $C_1 = \{4, 9\}$, $C_2 = \{9, 12\}$, $C_3 = \{4, 14\}$. C_1 and C_2 share block 9 but they are not connected to each other yet as shown in Figure 3.8(a). We can connect C_1 and C_2 by simply extending the length of C_1 and C_2 as shown in Figure 3.8(b). But there is a better solution in terms of bus length as shown in Figure 3.8(d). In our bus layout step, we will improve the solution in Figure 3.8(a) to a better solution as shown in Figure 3.8(d). The new solution after our bus layout step is shown in Figure 3.9.

To obtain the solution as shown in Figure 3.8(d), two steps are needed to be done. In the first step, some bus components may need to be moved first. Moving a bus component means that the y-coordinate of a horizontal bus component is

Algorithm 2 Bus Layout

1: **for** each bus net B_i **do**
2: /*move bus components*/
3: **for** each bus component C_j **do**
4: /*let C_j is a horizontal (vertical) bus component, $C_j = \{b_1, b_2, ..., b_n\}$ from left (up) to right (bottom)*/
5: **if** one of b_1 and b_n is on another vertical (horizontal) bus component C_k of B_i **then**
6: /* let b_1 be on another bus component and b_n is not*/
7: **if** b_1 is the uppermost (leftmost) / bottommost (rightmost) block of C_k **then**
8: move C_j to the bottommost (rightmost) / uppermost (leftmost) feasible position so as to shorten B_i as much as possible
9: **end if**
10: **else**
11: **if** b_1 is on other vertical (horizontal) bus components C_h of B_i and b_n is on other vertical (horizontal) bus components C_l of B_i **then**
12: **if** b_1 is the uppermost (leftmost) / bottommost (rightmost) of C_h and b_n is the uppermost (leftmost) / bottommost (rightmost) of C_l **then**
13: move C_j to the bottommost (rightmost) / uppermost (leftmost) feasible position
14: **end if**
15: **end if**
16: **end if**
17: **end for**/*move bus components*/
18: /*connecting bus components and remove redundancy*/
19: **for** each bus component C_j **do**
20: /*let C_j is a horizontal (vertical) bus component, $C_j = \{b_1, b_2, ..., b_n\}$ from left (up) to right (bottom)*/
21: **if** b_1 is on another vertical (horizontal) bus component C_k of B_i **then**
22: connect C_j to C_k
23: **else**
24: connect C_j to the right (bottom) boundary of b_1
25: **end if**
26: **if** b_n is on another vertical (horizontal) bus component $C_{k'}$ of B_i **then**
27: connect C_j to $C_{k'}$
28: **else**
29: connect C_j to the left (upper) boundary of b_n
30: **end if**
31: **end for**/*connecting bus components and remove redundancy*/
32: **end for**

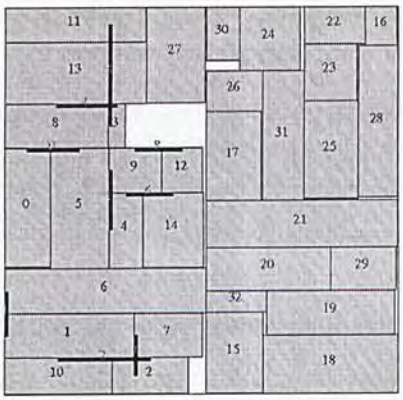

Figure 3.7: One bus routing solution $B_1 = \{0,5\}$, $B_2 = \{1,6\}$, $B_3 = \{2,7,10\}$, $B_4 = \{3,8,11,13\}$, $B_5 = \{4,9,12,14\}$

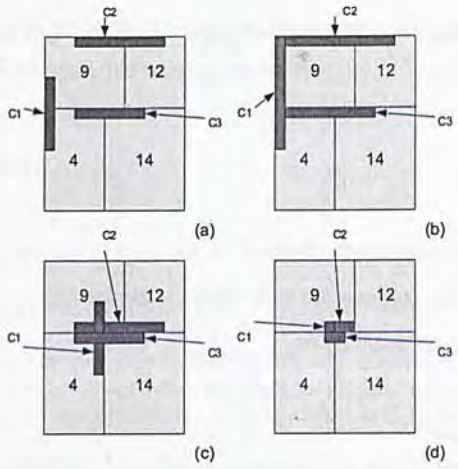

Figure 3.8: An example of bus layout $B_5 = \{4,9,12,14\}$

changed or the x-coordinate of a vertical bus component is changed. Bus components are moved towards the direction which results in a solution with shorter bus length. For a vertical bus component, if there are other bus components belonging to the same bus net on its right (left) side and there is no bus component belonging to the same bus net on its left (right) side, the bus component will be moved to its rightmost (leftmost) feasible position.(A position is called a *feasible position* for a

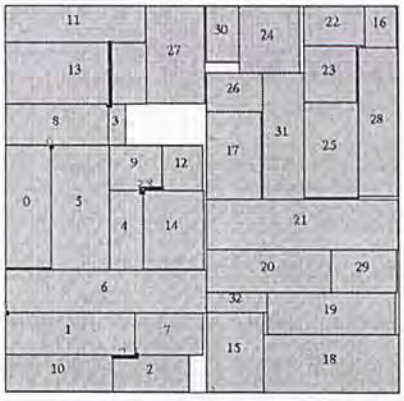

Figure 3.9: A new solution after bus layout step $B_1 = \{0,5\}$, $B_2 = \{1,6\}$, $B_3 = \{2,7,10\}$, $B_4 = \{3,8,11,13\}$, $B_5 = \{4,9,12,14\}$

bus component if the bus component does not overlap with other bus components at that position and the original bus topology is maintained.) Similarly, for a horizontal bus component, if there are other bus components belonging to the same bus net below (above) it and there is no bus component belonging to the same bus net above (below) it, the bus component will be moved to its lowest (uppermost) feasible position. Take bus B_5 as an example. It has three bus components $C_1 = \{4,9\}$, $C_2 = \{9,12\}$ and $C_3 = \{4,14\}$. To connect C_1 and C_2, the best choice is to move C_1 to the rightmost feasible position and move C_2 to the bottommost feasible position as shown in Figure 3.8(c).

In the second step, the redundant part of every bus component will be removed by connecting the bus component to the corresponding block boundary or to another bus component. Details are shown in Algorithm 2. If the block on the bus component C_i is not passed through by other bus components belonging to the same bus net, the bus component C_i is connected to the boundary of the block. Otherwise, if it is also passed through by another bus component C_j belonging to the same bus net, the bus component C_i is connected to C_j. In this example, we connect the right part of C_3 to the boundary of the corresponding block 14 and

connect the left part of C_3 to the bus component C_1. We also connect the right part of C_2 to the boundary of block 12 and connect the left part of C_2 to bus component C_1. We then connect the upper part of C_1 to C_2 and the lower part of C_1 to C_3. Finally, we get the new bus layout as shown in Figure 3.8(d). The position of a bus pin can be decided by finding the intersecting point of a bus component and the boundary of the corresponding block.

3.8 Experimental Results

Our approach is implemented using the C language. We run the floorplanner of [4] and our floorplanner on the same machine, an Intel Core2 Duo 2.33GHz CPU with 2GB memory. The floorplanner of [5] is run on a 1.86-GHz Linux machine with 2GB memory. The detailed information of all the benchmarks are shown in Table 3.1. The weights in Equation (3.1) are set automatically by random walk, so each test case is run ten times and the average is reported. We have performed two experiments. In the first experiment, we compare our result with the result of [4] and that of [5]. For fair comparison, we just minimize the bus length and total area in our floorplanner because the works in [4] and [5] just minimize bus length and total area without the fixed-outline constraint and the interconnect cost minimization. The buses are connected to the block boundaries in both [5] and our approach (Table 3.2). In [5], the position of each bus pin is not fixed at one single point on block boundary and the authors move the bus pin along the block boundary to reduce the total bus length. However, for fair comparison, when we compare our results with that of [4], we connect buses to the block centers because the floorplanner in [4] will connect buses to the block centers (Table 3.3). The results are shown in Table 3.2 and Table 3.3. The values after the slashes are the normalized values. The comparisons for all the data sets are reported in terms of bus length, running time and deadspace. In [5], no results are reported for bench-

Table 3.1: Data Set

Data	No. of Blocks	No. of Buses	Average/Max. No. of Blocks on a Bus Net
ami33-1	33	8	4.17 / 6
ami33-2	33	18	2.39 / 4
ami33-3	33	1	10 / 10
ami33-4	33	3	10 / 10
ami33-5	33	5	10 / 10
ami33-a	33	5	3 / 4
ami33-b	33	5	4 / 5
ami33-c	33	5	5 / 6
ami33-d	33	5	6 / 7
ami33-e	33	5	7 / 8
ami33-f	33	5	8 / 9
ami49-1	49	9	4.00 / 6
ami49-2	49	12	3.58 / 6
ami49-3	49	15	3.53 / 6
ami49-4	49	1	15 / 15
ami49-5	49	3	11.67 / 15
ami49-6	49	4	11.25/ 15
ami49-a	49	1	10
ami49-b	49	1	20
ami49-c	49	1	30
ami49-d	49	1	40
ami49-e	49	1	49

mark ami33-1 to ami33-5 and ami49-1 to ami49-6, so we do not compare those results. Compared with [5], their bus length is 1.28X of ours and their deadspace is 1.03X of ours on average. The two floorplanners are run on different machines, so we do not compare the running time. Compared with [4], their bus length is 1.32X of ours, their running time is 2.42X of ours and their deadspace is 1.43X of ours on average. Our bus length in Table 3.3 is much different from that in Table 3.2 for two reasons. Firstly, we connect buses to the block boundaries in

Table 3.2: Comparisons on Bus Length, Running Time and DeadSpace between Fixed-outline BDF and [5]

Test Cases	Fixed-outline BDF			[5]		
	Bus Length*	Running Time (sec)	Deadspace (%)	Bus Length*	Running Time (sec)	Deadspace (%)
ami33-a	550.45 / 1	5.64	1.75 / 1	710.50 / 1.29	4.11	1.56 / 0.89
ami33-b	986.09 / 1	7.73	1.72 / 1	1366.70 / 1.39	6.13	1.67 / 0.97
ami33-c	1389.27 / 1	10.91	1.22 / 1	2433.20 / 1.75	7.35	1.72 / 1.41
ami33-d	1898.09 / 1	12.55	1.13 / 1	3318.40 / 1.75	8.05	1.87 / 1.65
ami33-e	2799.27 / 1	15.36	1.65 / 1	4578.60 / 1.64	8.91	1.88 / 1.14
ami33-f	2741.91 / 1	18.27	1.02 / 1	5078.60 / 1.85	10.01	1.98 / 1.94
ami49-a	5154.64 / 1	15.82	1.54 / 1	5304.40 / 1.03	8.96	0.86 / 0.56
ami49-b	11651.36 / 1	23.73	1.64 / 1	8189.40 / 0.70	10.14	0.95 / 0.58
ami49-c	16164.18 / 1	30.09	1.66 / 1	12764.60 / 0.79	12.84	1.3 / 0.78
ami49-d	19439.55 / 1	34.91	1.92 / 1	16955.80 / 0.87	16.96	1.56 / 0.81
ami49-e	22277.90 / 1	48.1	2.42 / 1	22452.60 / 1.01	21.86	1.5 / 0.62
Average	1	+	1	1.28		1.03

* The buses are connected to the block boundaries in both [5] and our approach.

\+ The results of [5] is from that paper, thus the two floorplanners are run on different machines and their running times are hard to compare.

Table 3.2 and connect buses to the block centers in Table 3.3. Connecting buses to the block boundaries and connecting buses to the block centers are two different issues. Secondly, our method is aimed to optimize the bus length connecting to the boundaries. Compared with other previous works, it is more accurate that we use the modified MST algorithm to generate the topology of a bus net when we estimate the bus length connecting to the boundaries. Therefore, our bus length in Table 3.2 is much shorter. The floorplanner of [4] cannot give a feasible solution after running ten times on the data sets ami49-c, ami49-d and ami49-e.

In the second experiment, we handle the fixed-outline constraint and minimize both the bus length and the interconnect cost. In our experiment, deadspace threshold r is set to 10% and the aspect ratio R of the chip is 1.0. Therefore the W and H are computed as $\sqrt{1.1 \times \sum A_i}$ where A_i is the area of block b_i. The aspect ratio bound of a soft block b_i is $[t_i - 0.8, t_i + 0.8]$ where t_i is the original aspect ratio of block b_i. We use HPWL to estimate wire length. The information of the pins and

Table 3.3: Comparisons on Bus Length, Running Time and DeadSpace between Fixed-outline BDF and [4]

Test Cases	Fixed-outline BDF			[4]		
	+ Bus Length*	Running Time (sec)	Deadspace (%)	Bus Length*	Running Time (sec)	Deadspace (%)
ami33-1	4826.55 / 1	15.36 / 1	1.70 / 1	6589.17 / 1.37	21.67 / 1.41	2.13 / 1.25
ami33-2	4539.36 / 1	19.18 / 1	1.92 / 1	8684.33 / 1.91	20.17 / 1.05	2.18 / 1.14
ami33-3	1436.91 / 1	4.45 / 1	1.37 / 1	1913.17 / 1.33	9.67 / 2.17	1.16 / 0.85
ami33-4	4242.27 / 1	9.91 / 1	1.88 / 1	5158.00 / 1.22	39.17 / 3.95	4.03 / 2.14
ami33-5	8625.00 / 1	18.36 / 1	5.16 / 1	9458.33 / 1.10	78.67 / 4.28	4.53 / 0.88
ami33-a	1835.55 / 1	5.64 / 1	0.90 / 1	3381.00 / 1.84	9.33 / 1.65	2.09 / 2.32
ami33-b	2737.18 / 1	7.18 / 1	1.18 / 1	4882.17 / 1.78	11.00 / 1.53	1.78 / 1.51
ami33-c	3552.64 / 1	10.27 / 1	1.31 / 1	5124.83 / 1.44	13.33 / 1.30	1.58 / 1.21
ami33-d	4439.73 / 1	12.00 / 1	1.43 / 1	5886.00 / 1.33	22.50 / 1.88	1.48 / 1.03
ami33-e	5067.91 / 1	14.73 / 1	2.03 / 1	5903.17 / 1.16	69.00 / 4.68	4.76 / 2.34
ami33-f	5579.55 / 1	17.27 / 1	1.23 / 1	5942.00 / 1.06	33.00 / 1.91	2.91 / 2.37
ami49-1	30778.55 / 1	50.73 / 1	1.91 / 1	39218.80 / 1.27	62.40 / 1.23	2.44 / 1.28
ami49-2	41833.55 / 1	85.91 / 1	2.37 / 1	48009.17 / 1.15	96.17 / 1.12	2.27 / 0.96
ami49-3	47215.20 / 1	100.50 / 1	4.68 / 1	54117.00 / 1.15	110.40 / 1.10	2.28 / 0.49
ami49-4	12861.50 / 1	15.80 / 1	1.20 / 1	13178.83 / 1.02	31.50 / 1.99	1.61 / 1.34
ami49-5	24315.36 / 1	28.00 / 1	1.70 / 1	32069.00 / 1.32	90.00 / 3.21	2.06 / 1.21
ami49-6	29647.36 / 1	44.45 / 1	1.56 / 1	43057.00 / 1.45	347.33 / 7.81	3.49 / 2.24
ami49-a	10263.80 / 1	13.10 / 1	1.25 / 1	11778.83 / 1.15	30.33 / 2.32	1.54 / 1.23
ami49-b	15641.27 / 1	19.73 / 1	1.34 / 1	15727.50 / 1.01	28.50 / 1.44	1.78 / 1.33
ami49-c	20276.82 / 1	26.82 / 1	1.68 / 1	NA** /	NA /	NA /
ami49-d	25591.18 / 1	30.45 / 1	1.39 / 1	NA /	NA /	NA /
ami49-e	31295.00 / 1	43.00 / 1	1.68 / 1	NA /	NA /	NA /
Average	1	1	1	1.32	2.42	1.43

* We connect buses to the block centers because the floorplanner in [4] just connects buses to the block centers.

** The floorplanner of [4] cannot give a feasible solution after running ten times on the data sets ami49-c, ami49-d and ami49-e.

netlist is shown in Table 3.5. We compare our approach with a modified version of the floorplanner in [4][2]. In this modified floorplanner, we add the fixed-outline post-processing step and consider interconnect minimization in the objective function of the annealing process. We also use our bus layout step presented in Section 3.7 to connect buses to the block boundaries at the end. Experimental results are shown in Table 3.4. The results for all the data sets are reported in terms of

[2]We choose to modify the floorplanner in [4] for two reasons. Firstly, we have the source code of the floorplanner in [4]. Secondly, the flow of the floorplanner in [4] is similar to ours.

bus length, wire length and running time. Experimental results show that the bus length of the modified version of [4] is on average 2.03X of our bus length and their wire length is on average almost the same as ours. This modified floorplanner of [4] cannot give a feasible solution after running ten times on the data sets ami49-d and ami49-e. A floorplanning solution is shown in Figure 3.10. The complexity and run time of our method do not increase with an increasing number of metal layers.

Table 3.4: Comparisons on Bus Length, Running Time and Wire Length between Fixed-outline BDF and Modified Floorplanner of [4]

Test Cases	Fixed-outline BDF			Modified Floorplanner of [4]		
	Bus Length*	Running Time (sec)	Wire Length	Bus Length*	Running Time (sec)	Wire Length
ami33-1	2593.50 / 1	15.12 / 1	88721.62 / 1	5295.20 / 2.04	24.80 / 1.64	89721.00 / 1.01
ami33-2	1132.14 / 1	22.86 / 1	87674.00 / 1	4996.00 / 4.41	22.50 / 0.98	91004.00 / 1.04
ami33-3	1282.64 / 1	4.55 / 1	81766.91 / 1	1875.10 / 1.46	6.90 / 1.52	88012.90 / 1.08
ami33-4	3427.00 / 1	10.71 / 1	85935.29 / 1	5233.86 / 1.53	17.71 / 1.65	92861.29 / 1.08
ami33-5	5978.75 / 1	22.50 / 1	96991.88 / 1	8855.00 / 1.48	33.00 / 1.47	97217.00 / 1.00
ami33-a	1041.89 / 1	6.44 / 1	88626.33 / 1	2151.89 / 2.07	8.56 / 1.33	84692.56 / 0.96
ami33-b	1435.33 / 1	9.11 / 1	90474.78 / 1	3823.80 / 2.66	11.70 / 1.28	86888.60 / 0.96
ami33-c	1704.44 / 1	12.56 / 1	93100.78 / 1	4223.40 / 2.48	16.70 / 1.33	92181.50 / 0.99
ami33-d	2446.00 / 1	13.75 / 1	95060.38 / 1	4864.71 / 1.99	21.43 / 1.56	95789.00 / 1.01
ami33-e	3276.44 / 1	16.56 / 1	99669.44 / 1	5875.86 / 1.79	28.14 / 1.70	95498.86 / 0.96
ami33-f	3291.91 / 1	20.91 / 1	76746.64 / 1	6473.86 / 1.97	23.00 / 1.10	91879.43 / 1.20
ami49-1	13395.00 / 1	33.00 / 1	1515660.00 / 1	34523.50 / 2.58	79.67 / 2.41	1452900.67 / 0.96
ami49-2	14060.67 / 1	54.67 / 1	1602456.00 / 1	38163.40 / 2.71	97.40 / 1.78	1430490.20 / 0.89
ami49-3	18584.83 / 1	58.67 / 1	1622330.67 / 1	43292.33 / 2.33	191.00 / 3.26	1324948.33 / 0.82
ami49-4	12044.78 / 1	17.78 / 1	1207856.00 / 1	12633.50 / 1.05	32.83 / 1.85	1204466.50 / 1.00
ami49-5	22143.33 / 1	34.56 / 1	1163613.11 / 1	28692.00 / 1.30	72.50 / 2.10	1228114.00 / 1.06
ami49-6	22879.18 / 1	42.45 / 1	1102990.64 / 1	38318.00 / 1.67	97.50 / 2.30	1303445.50 / 1.18
ami49-a	4511.33 / 1	16.67 / 1	1086896.00 / 1	10066.30 / 2.23	35.40 / 2.12	1240945.60 / 1.14
ami49-b	9609.00 / 1	41.33 / 1	1150908.33 / 1	16498.50 / 1.72	36.25 / 0.88	1285767.00 / 1.12
ami49-c	18672.20 / 1	22.00 / 1	1381215.00 / 1	19302.00 / 1.03	61.00 / 2.77	1103986.00 / 0.80
ami49-d	19888.00 / 1	20.00 / 1	1459632.00 / 1	NA1 /	NA /	NA /
ami49-e	27090.00 / 1	40.00 / 1	1396458.00 / 1	NA /	NA /	NA /
Average	1	1	1	2.03	1.75	1.01

* The buses are connected to the block boundaries in both our floorplanner and the modified floorplanner of [4].

) This modified floorplanner of paper [4] cannot give a feasible solution after running ten times on the data sets ami49-d and ami49-e.

Table 3.5: Pins and NetList Information

Test Cases	No. of Blocks	No. of Pins	No. of Nets
ami33-x	33	42	123
ami49-x	49	22	408

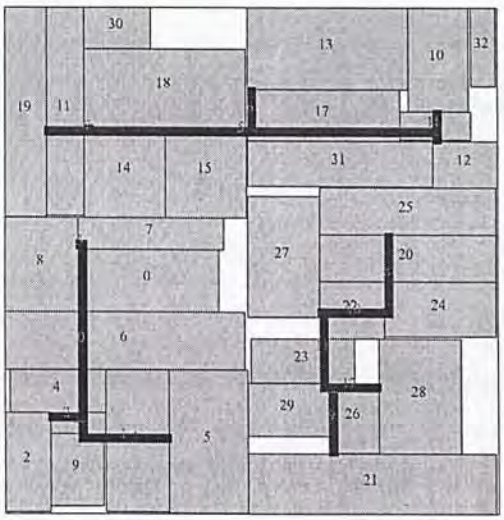

Figure 3.10: A bus routing solution $B_1 = \{0,1,2,3,4,5,6,7,8,9\}$, $B_2 = \{10,11,12,13,14,15,16,17,18,19\}$, $B_3 = \{20,21,22,23,24,25,26,27,28,29\}$

3.9 Summary

In this chapter, we presented a floorplanner that can give a fixed-outline floorplanning solution including bus route that minimizes bus length and interconnect cost. We proposed two steps to minimize bus length. Firstly, we devised a bus-driven partitioning step to minimize the total bus length. Recursive bi-partitioning is applied at the beginning to handle the problem globally. Secondly, we propose a modified MST construction algorithm to generate a bus topology to reduce bus length. Experimental results show that we can improve over the previous work [4]

in terms of both bus length, running time and deadspace. Besides, we also improve over another recent work [5] in terms of bus length on average.

□ **End of chapter.**

Chapter 4

Fixed-Outline BDF with L-shape bus

4.1 Introduction

For some sequence pairs, there are no feasible solutions if only zero-bend buses, one-bend buses and two-bend buses are allowed or if the turning points of the buses are only allowed to occur on the blocks on the bus net. Some bus components may have conflicting block alignment constraints. Let's see an example. Suppose two bus components $B_1 = \{b_1, b_4\}$ and $B_2 = \{b_2, b_3\}$. The sequence pair is $\{(b_1, b_2, b_3, b_4), (b_2, b_1, b_4, b_3)\}$. B_1 and B_2 are horizontal bus components but only one of them can be routed successfully by block alignment because b_2 and b_3 cannot be aligned if b_1 and b_4 are aligned or vice versa in this scenario.

In this chapter, bus routability problem is handled in a more sophisticated manner. We propose a method to insert L-shape bus components. Our floorplanner can give a fixed-outline floorplanning solution including bus routes that minimizes bus length, number of extra vias and interconnect cost. In each iteration of the simulated annealing process, our goal is to route all the bus components successfully by inserting L-shape bus components. We first present a method to detect conflicting block alignment constraints and select some bus components to be converted into

L-shape bus components to make sure that there is no conflicting block alignment constraint. The selected bus components are then routed in L-shape according to the method described in Section 4.3.1. In addition, some bus components can be combined to reduce the number of bus components. We propose a method to combine bus components to reduce the number of bus components.

Experimental results show that we also improve over the recent work [5] in terms of bus length and successful rate on average. Besides, we can improve over our first floorplanner in chapter 3 and modified floorplanner of [4] in terms of both bus length, running time and successful rate.

4.2 Problem Formulation

The problem we consider in this chapter is similar to that in chapter 3 except that the bendings of the buses are allowed to occur on non-bus blocks. We are given the following:

1. A set of blocks and each block has an area and an aspect ratio bound
2. A set of buses and each bus has a width and a bus net
3. A set of nets
4. A set of pins and each pin has coordinates
5. Fixed outline $W \times H$

The aim is to obtain a fixed-outline floorplan solution including bus routes that minimizes the bus length, number of extra vias and interconnect cost. We need to decide the position of each block and layout the buses on two metal layers, one horizontal and one vertical. No overlapping between any two blocks and between any two buses on the same layer are allowed. Besides, the bendings of the buses

can occur on non-bus blocks. For the example in Figure 3.1, the possible BDF solution in Figure 3.1(A) is also a feasible solution.

4.3 Our Approach

An overview of our approach is given in Figure 4.1. Our approach is composed of four steps. The first step is partitioning that is presented in Section 3.4. Blocks are partitioned into several groups and the result of this partitioning step is used as the initial candidate solution of the next floorplanning step. The second step is a SA process. The floorplanning step is done by SA. The objective of this step is to find a floorplan solution with feasible bus routing that minimizes the bus length, chip area, number of extra vias and wire length. In each iteration of this process, the topology of each bus is generated by the method described in Section 3.5.1. According to the bus topology generated, we obtain several bus components. The bus routability is checked to make sure that all the bus components can be routed successfully. If some bus components have conflicting block alignment constraints, a set of bus components will be selected to be converted into L-shape bus components. To reduce the total number of bus components, some bus components will then be combined to form new bus components. Detailed discussion will be given in Section 4.3.1. The bus routing constraints (i.e. aligning blocks to allow buses to pass through and assigning bus ordering to avoid bus overlapping) are handled by adding a dummy block and some constraint edges into the horizontal constraint graph G_h or the vertical constraint graph G_v for each bus component in the new set of bus components. Details are presented in Section 3.5.2. Finally, the coordinates of the blocks and the bus components are calculated by performing a single source longest path algorithm in G_h and G_v. In the third step, another annealing process described in Section 3.6 is used to handle the fixed-outline constraint based on the solution obtained in the second step. Finally, we layout the buses and decide on

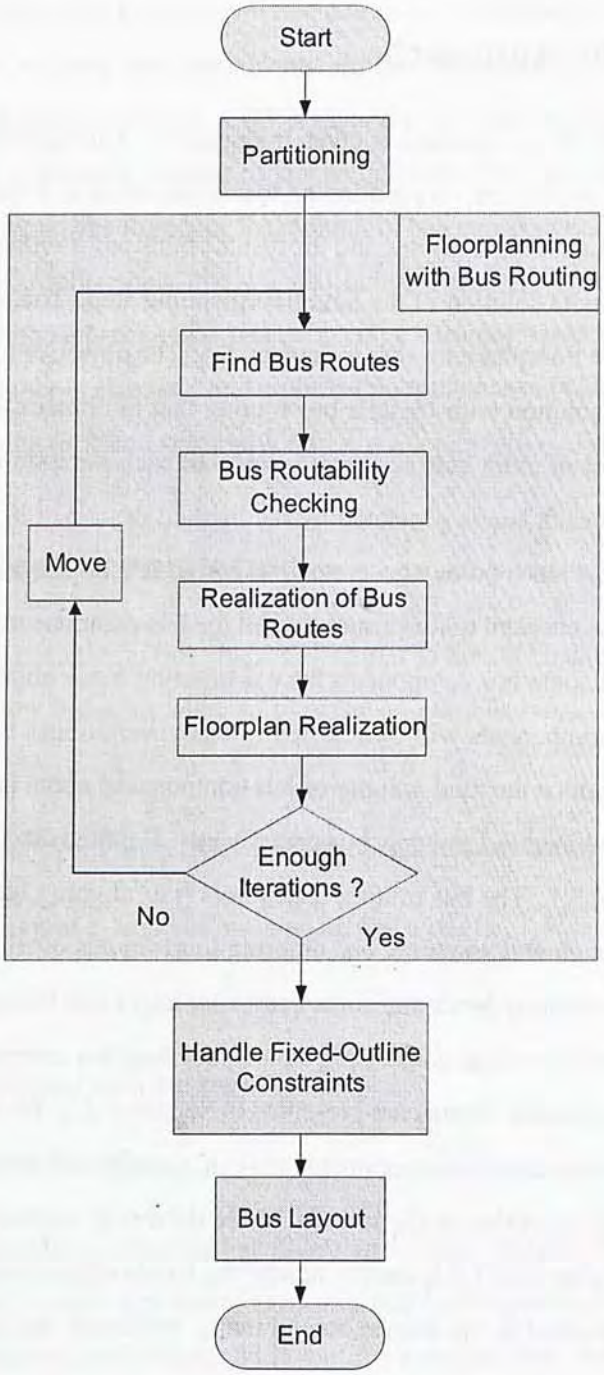

Figure 4.1: Overview of our second approach

the positions of the bus pins. Details are discussed in Section 3.7.

4.3.1 Bus Routability Checking

Some bus components may have conflicting block alignment constraints. In our method, some bus components are selected to be divided into an L-shape bus component. An *L-shape bus component* contains two bus components, one is horizontal and the other is vertical. We add a virtual block with width t and height t (t is the width of the bus) for the intersection of these two bus components. Take the example described in Section 4.1, in Figure 4.2, B_1 is divided into an L-shape bus component. In this way, B_1 and B_2 can be routed successfully although there is an extra via for the turning point of the bus. However, the number of extra vias will be minimized in our objective function of the SA process.

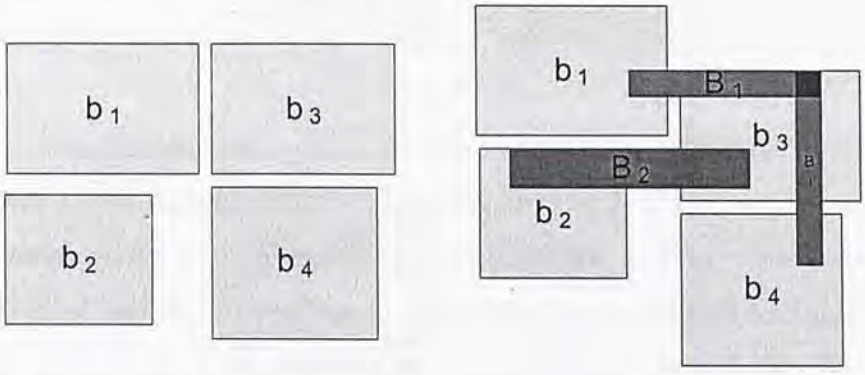

Figure 4.2: An L-shape bus for conflicting block alignment constraints

Therefore, the bus routability needs to be checked to see if all the bus components can be routed successfully. In the bus routability checking process, three steps will be done. The first step is to detect violations of block alignment constraints. After this step, some bus components are selected to be converted into L-shape. The second step is to convert those bus components into L-shape and to sort all the bus components. The horizontal bus components will be sorted such

that a horizontal bus component B_i is in front of another horizontal bus component B_j if none of the blocks in B_i is above any of the blocks in B_j. The vertical bus components are also sorted such that a vertical bus component B_i is in front of another vertical bus component B_j if none of the blocks in B_i is on the right hand side of any of the blocks in B_j. The third step is to combine some bus components to reduce the total number of bus components. After this process, we get a new set of bus components. An overview of this process is given in Figure 4.3.

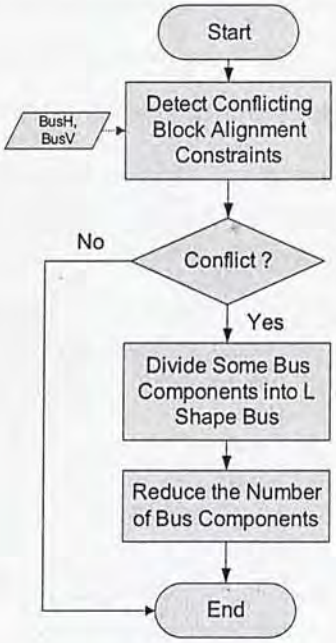

Figure 4.3: Overview of bus routability checking

Detect Violations of Block Alignment Constraints

After we construct a MST for each bus net, we get two sets of bus components, a set of horizontal bus components *BusH* and a set of vertical bus component *BusV*. Each bus component connects two blocks. For some sequence pairs, the block alignment constraints for some bus components are conflicting with each other. In

this scenario, some bus components will be selected to be converted into L-shape.

In our method, to detect violations of block alignment constraints, two *bus constraint graphs* (horizontal bus constraint graph and vertical bus constraint graph) are constructed according to the relationship of the blocks in different bus components. Take the horizontal bus constraint graph as an example. Each horizontal bus component B_i in *busH* is represented as a vertex v_i in the horizontal bus constraint graph. For any two horizontal bus components $B_i = \{b_k, b_l\}$ and $B_j = \{b_{k'}, b_{l'}\}$, we will insert an edge from v_j and v_i in the horizontal bus constraint graph if and only if there is any block of B_i above any block of B_j. The pseudo code of the method to construct the horizontal bus constraint graph is given in Algorithm 3. The construction of the vertical bus constraint graph can be done similarly.

If there are cycles in the graph, it means that some block alignment constraints for bus routing are conflicting with each other. The cycles can be detected by checking if the in-degree of each vertex is larger than zero. If all the vertices' in-degrees are larger than zero, there are cycles in the graph. We need to select some bus components and convert them into L-shape bus components.The detailed method is given as follows. The vertices with zero in-degree will be removed from the horizontal bus constraint graph. The edges incident on those vertices are also removed. These two steps are repeated until there is no vertex with zero in-degree. If this occurs before all vertices are removed, it means that there is at least one cycle in the horizontal bus constraint graph. The vertex with the maximum degree will then be deleted from the graph and all edges incident on this vertex will be deleted until there is at least one vertex with zero in-degree in the graph. The corresponding bus components of those deleted vertices (with maximum degree) will be selected to be divided into L-shape bus components. The detailed method will be discussed in next section. An *initial horizontal/vertical sorted bus component list* is constructed when bus components with zero in-degree are removed. For ex-

Algorithm 3 Construct horizontal bus constraint graph

```
1: for each horizontal bus component B_i do
2:    add one vertex v_i in the horizontal bus constraint graph
3: end for
4: for any two horizontal bus components B_i and B_i' do
5:    flag1 = 0 and flag2 = 0
6:    for each block b_j in bus component B_i do
7:       for each block b_j' in bus component B_i' do
8:          if b_j is above b_j' then
9:             flag1 = 1
10:         else
11:            if b_j' is above b_j then
12:               flag2 = 1
13:            end if
14:         end if
15:      end for
16:   end for
17:   if flag1 = 1 then
18:      add an edge from v_i' to v_i in the horizontal bus constraint graph
19:   end if
20:   if flag2 = 1 then
21:      add an edge from v_i to v_i' in the horizontal bus constraint graph
22:   end if
23: end for
```

ample, if we remove three horizontal bus components in the order B_2, B_1, B_3, the initial horizontal sorted bus component list is B_2, B_1, B_3. After all the vertices are removed from the graph, we get an ordering of those bus components that are not

converted into L-shape.

Insert L-Shape Bus Component

As we discussed before, an L-shape bus component contains two bus components. One is horizontal and the other is vertical. We will add a virtual block with width t and height t (t is the width of the bus) for the intersection of these two bus components. The use of L-shape buses will increase the number of extra vias but it will be minimized in our objective function of the SA process. There are two kinds of L-shape bus components as shown in Figure 4.4.

The L-shape bus component is different from the 1-bend bus in [3]. For a bus connecting two blocks, either a horizontal bus or a vertical bus between the two blocks is allowed in [3]. However, in our method three solutions are allowed, a horizontal bus, a vertical bus or L-shape bus as shown in Figure 3.1(A). More sophisticated shapes are not allowed in this method because they will introduce more vias.

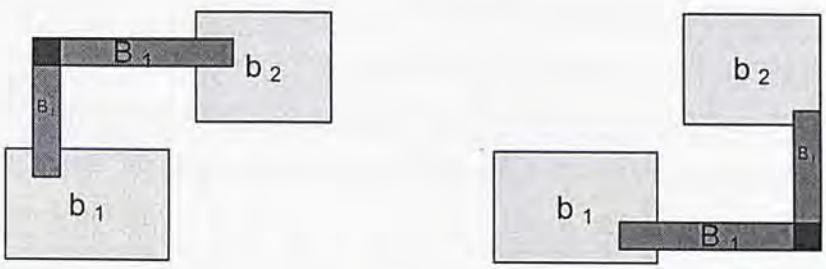

Figure 4.4: Two kinds of L-shape bus

After all the bus components are divided into L-shape bus components, two sets of bus components from L-shape bus components are generated, the set of horizontal bus components BL_h and the set of vertical bus components BL_v. All the bus components can be routed after certain bus components are divided to L-shape bus components since there is no conflicting block alignment constraint.

Construction of Sorted Bus Component List

To check if the number of bus components can be reduced, a structure called *sorted bus component list* will be constructed according to the relative position of the blocks on each bus component. A sorted bus component list is a list of bus components sorted according to their blocks' horizontal constraints or vertical constraints. Two sorted bus component lists are generated. For horizontal bus components, a *horizontal sorted bus component list* is generated by sorting the horizontal bus components according to their blocks' vertical constraints. The horizontal sorted bus component list is constructed based on the initial horizontal sorted bus component list generated in the first step. Note that the initial horizontal sorted bus component list does not include the bus components that need to be converted into L-shape. If there are bus components that need to be converted into L-shape, the L-shape bus components are then inserted into the initial sorted bus component list according to their blocks' horizontal constraints or vertical constraints. Each bus component B_i in the set of horizontal bus components BL_h for L-shape bus components is inserted into the initial horizontal sorted bus component list. The pseudo code of constructing horizontal sorted bus component list is given in Algorithm 4. Note that there will be no conflict resulted because all the conflicting constraints are detected and solved by converting some bus components into L-shape in the step of detecting conflicting block alignment constraints. At the end, the horizontal sorted bus component list is constructed. For vertical bus components, a *vertical sorted bus component list* can also be generated by performing the similar operations.

Reduce the Number of Bus Components

After we detect conflicting block alignment constraints and convert some bus components into L-shape, four sets of bus components are generated. Some bus com-

Algorithm 4 Construct horizontal sorted bus component list

```
1: for each bus component B_i in BL_h do
2:     for each bus component B_j in the initial horizontal sorted bus component
       list do
3:         if there exists one block related to B_j above one block related to B_i then
4:             insert B_i at the position before B_j in the initial horizontal sorted bus
               component list and break
5:         end if
6:     end for
7: end for
```

ponents will be combined to reduce the total number of bus components. For example, consider two buses $B = \{b_1, b_4, b_5\}$ and $B' = \{b_2, b_3\}$. Assume that the sequence pair is $\{(b_1, b_5, b_2, b_3, b_4), (b_2, b_1, b_5, b_4, b_3)\}$. B and B' are horizontal buses. Suppose that we have three bus components $B_1 = \{b_4, b_5\}$, $B_2 = \{b_2, b_3\}$ and $B_3 = \{b_1, b_5\}$ from the step described in Section 3.5.1. Only one of B_1 and B_2 can be routed successfully because b_2 and b_3 cannot be aligned if b_4 and b_5 are aligned or vice versa. Suppose that B_1 is divided into L-shape bus component. Then we have three horizontal bus components and one vertical bus component as shown in Figure 4.5. We can combine B_3 and the horizontal part of B_1 into a new bus component B_4 to reduce the number of horizontal bus components from 3 to 2.

However, some bus components cannot be combined as shown in Figure 4.6. In this example, the sequence pair is $\{(b_1, b_4, b_2, b_5, b_3), (b_4, b_1, b_2, b_3, b_5)\}$. There are two bus nets $B = \{b_1, b_2, b_3\}$ and $B' = \{b_4, b_5\}$. Suppose that we have three horizontal bus components from MST. $B_1 = \{b_4, b_5\}$, $B_2 = \{b_1, b_2\}$ and $B_3 = \{b_2, b_3\}$. We cannot combine B_2 and B_3 although they belong to the same bus net B. If we combine them, there will be conflicting block alignment because b_1 and

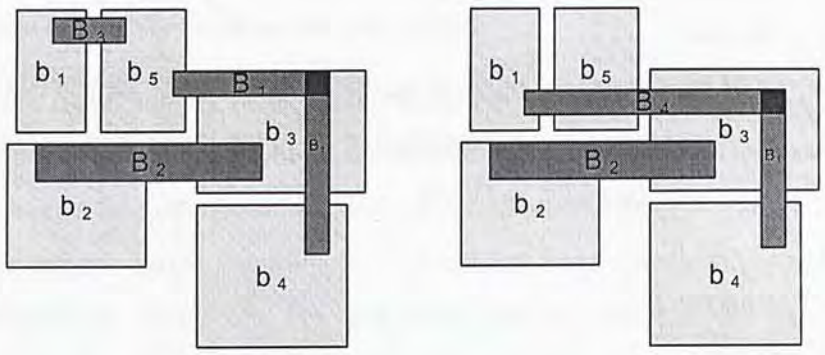

Figure 4.5: One example of combining two horizontal bus components

b_3 cannot be aligned if b_4 and b_5 are aligned or vice versa.

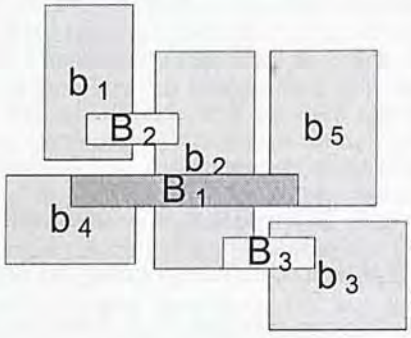

Figure 4.6: Another example of combining two horizontal bus components

A set of bus components (two or more bus components) satisfying the following five conditions can be combined to a new bus component.

C1.1 The bus components belong to the same bus net.

C1.2 All the bus components are of the same type, such as, all are horizontal bus components or all are vertical bus components.

C1.3 For any bus component B_i in the set, there exists another bus component $B_{i'}$ in the set sharing one block with it. For example, $B_1 = \{b_i, b_j\}$ and

$B_2 = \{b_j, b_k\}$ are two horizontal bus components that belong to the same bus net and share one block b_j.

C1.4 For horizontal bus components, there is no vertical constraint between blocks on different bus components in the set. For vertical bus components, there is no horizontal constraint between blocks on different bus components in the set.

C1.5 There should be no cycle formed in the bus constraint graph if we delete B_1, B_2 and add a new bus component B_3 where B_3 is formed by combining B_1, B_2 such that the edges incident on B_1 and B_2 are combined to form the edges incident on B_3 in the bus constraint graph.

To check if two bus components can be combined, the above five conditions should be checked. We can check if two bus components satisfy conditions $C1.1$ to $C1.4$ easily. We can then check if those bus components satisfying conditions $C1.1$ to $C1.4$ satisfy condition $C1.5$ by checking the bus components' indices in the sorted bus component list. Two bus components B_i and B_j satisfying conditions $C1.1$ to $C1.4$ will satisfy condition $C1.5$ (can be combined) if they satisfy one of the following two conditions: Suppose that the index of B_i in the sorted bus component list is x and the index of B_j in the sorted bus component list is y and $x < y$.

C2.1 $y - x = 1$ or

C2.2 B_i can be swapped with the bus components with bigger indices (the definition of swapping two bus components will be given in the next paragraph) while B_j can be swapped with the bus components with smaller indices such that B_i's new index x' and B_j's new index y' satisfy $|x' - y'| = 1$

For horizontal bus components, a bus component B_i can be swapped with another bus component B_j in the horizontal sorted bus component list if and only if there

is no vertical constraint between the blocks on B_i and the blocks on B_j. Similarly for vertical bus components, B_i can be swapped with B_j in the vertical sorted bus component list if and only if there is no horizontal constraint between the blocks on B_i and the blocks on B_j.

For any two horizontal bus components B_i and B_j, if their indices x and y satisfy condition $C2.1$, they can be combined to form a new bus component B_i'. The reason is that the bus components below B_i can be routed below B_i' and the bus components above B_j can be routed above B_i'. That means, the new bus component B_i' have no conflicting block alignment constraint with other bus components. Thus the two horizontal bus components B_i and B_j can be combined to form a new bus component. Similarly, for any two horizontal bus components B_i and B_j, if their indices x and y satisfy condition $C2.2$, they can also be combined to form a new bus component B_i'. Since we only swap two horizontal bus components if and only if there is no horizontal constraint between the blocks on B_i and the blocks on B_j, there is no conflicting block alignment constraint if the two bus components are combined to form a new one.

Suppose that the horizontal sorted bus component list is $B_1, B_2, \ldots, B_n$ and we want to check if B_x and B_y can be combined to form a new bus component according to conditions $C2.1$ and $C2.2$. Let x,y be the indices of the bus components in the horizontal sorted bus component list and they satisfy $x < y$. The pseudo code is given in Algorithm 5.

If two bus components B_i and B_j satisfy condition $C2.1$ or $C2.2$, they are combined to form a new bus component B_i'. The corresponding sorted bus component list is then updated by deleting B_i and B_j and adding B_i' to the final position of B_i. The indices of all the other bus components are also updated.

If three or more bus components satisfy conditions $C1.1$ to $C1.4$, there are two possibilities for combining bus components. The first option is that all the bus

Algorithm 5 Checking if two horizontal bus components B_x and B_y can be combined (Suppose that the index of B_x in the sorted bus component list is x and the index of B_y in the sorted bus component list is y and $x < y$.)

```
1: if y − x = 1 then
2:     return B_x and B_y can be combined (satisfying C2.1)
3: else
4:     flag1 = 0 and flag2 = 0
5:     while y > x + 1 do
6:         if flag1 = 0 then
7:             if B_x and B_{x+1} can be swapped then
8:                 x = x + 1
9:             else
10:                flag1 = 1
11:            end if
12:        end if
13:        if flag2 = 0 then
14:            if B_y and B_{y−1} can be swapped then
15:                y = y − 1
16:            else
17:                flag2 = 1
18:            end if
19:        end if
20:        if flag1 = 1 and flag2 =1 then
21:            return B_x and B_y cannot be combined
22:        end if
23:        x' = x and y' = y
24:    end while
25:    if y' − x' = 1 then
26:        return B_x and B_y can be combined (satisfying C2.2)
27:    else
28:        return B_x and B_y cannot be combined
29:    end if
30: end if
```

components in the set will be combined to form a new bus component. The second option is that some of them will be combined and several new bus components are formed at the end.

In our method, if three or more bus components satisfy conditions $C1.1$ to $C1.4$, two bus components are checked every time to see if they satisfy $C2.1$ or $C2.2$ by using the Algorithm 5. For example, if we want to check if the bus components $B_1, B_2, \ldots, B_m$ (sorted according to their indices in the horizontal sorted bus component list) can be combined, the checking process starts from checking B_1 and B_2. If the two bus components can be combined, they will be combined to form a new bus component B_1'. Then, the new bus component B_1' and B_3 are checked. If B_1 and B_2 cannot be combined, B_1 forms a new bus component B_1'. We continue to check B_2 and B_3. The checking process is stopped until all the bus components are checked. At the end, we can obtain all the new bus components B_j' ($j \leq m$). The pseudo code is given in Algorithm 6.

Algorithm 6 Combine a set of bus components

```
1: j = 1 and B'_j = B_j
2: for each bus component B_i after (except B_1) do
3:    if B_i and B'_j can be combined then
4:       update B'_j (using B'_j = B'_j ∪ B_i)
5:       update index of each bus component
6:    else
7:       j = j + 1
8:       update B'_j (using B'_j := B_i)
9:    end if
10: end for
```

4.3.2 Details of the Annealing Process

In the SA process, the objective is to find a floorplan solution with feasible bus assignment and to minimize the bus length, total area and wire length.

Cost function

In the annealing process, we want to minimize the total area, bus length and wire length with a feasible bus connection. The cost function is defined as:

$$Cost = \alpha Cost_{Area} + \beta Cost_{bus} + \gamma Cost_{wire} + \delta Cost_p \quad (4.1)$$

where $Cost_{Area}, Cost_{bus}, Cost_{wire}$ are the total area, total bus length and total wire length of the floorplan solution respectively. The wire length is estimated with HPWL. $Cost_p$ includes the penalty for solutions with extra vias and it is the number of extra vias. The parameters α, β, γ and δ are weights obtained by performing random walk at the beginning of the annealing process. With this cost function, we minimize the total area of the packing first, and the fixed-outline constraints will be handled by a post-processing step as described in Section 3.6.

4.4 Experimental Results

Our approach is implemented using the C language. We compare our results with the results of our first floorplanner described in the previous chapter, and the results of [5]. We run our first floorplanner and our second floorplanner on the same machine, an Intel Core2 Duo 2.33GHz CPU with 2GB memory. The floorplanner of [5] is run on a 1.86-GHz Linux machine with 2GB memory. The weights in Equation (4.1) are set automatically by random walk, so each test case is run ten times and the average is reported. We have performed two experiments. In the first experiment, we compare our results with the results of our first floorplanner and

Table 4.1: Comparisons on Bus Length, Running Time, DeadSpace and Successful Rate between Fixed-outline BDF, Fixed-outline BDF with L-shape Bus and [5]

Test Cases	Fixed-outline BDF***				Fixed-outline BDF with L-shape bus				[5]			
	Bus Length*	Running Time (sec)	Dead Space (%)	Successful Rate** (%)	Bus Length*	Running Time (sec)	Dead Space (%)	Successful Rate** (%)	Bus Length*	Running Time (sec)	Dead Space (%)	Successful Rate** (%)
ami33-a	550.45 / 1.16***	5.64 / 0.99	1.75 / 1.86	100 / 1	474.5 / 1	5.67 / 1	0.94 / 1	100 / 1	710.5 / 1.5	4.11	1.56 / 1.66	99.8 / 1
ami33-b	986.09 / 1.18	7.73 / 1.19	1.72 / 0.86	100 / 1	835.5 / 1	6.5 / 1	1.99 / 1	100 / 1	1366.7 / 1.64	6.13	1.67 / 0.84	99.28 / 0.99
ami33-c	1389.27 / 0.84	10.91 / 1.65	1.22 / 0.76	100 / 1	1649.5 / 1	6.6 / 1	1.6 / 1	100 / 1	2433.2 / 1.48	7.35	1.72 / 1.08	98.18 / 0.98
ami33-d	1898.09 / 0.97	12.55 / 1.65	1.13 / 0.89	99.45 / 0.99	1949.1 / 1	7.6 / 1	1.27 / 1	100 / 1	3318.4 / 1.7	8.05	1.87 / 1.47	97.81 / 0.98
ami33-e	2799.27 / 1.08	15.36 / 1.9	1.65 / 0.64	99.49 / 0.99	2601.1 / 1	8.1 / 1	2.57 / 1	100 / 1	4578.6 / 1.76	8.91	1.88 / 0.73	95.46 / 0.95
ami33-f	2741.91 / 0.94	18.27 / 1.66	1.02 / 0.49	98.72 / 0.99	2921.6 / 1	11 / 1	2.08 / 1	100 / 1	5078.6 / 1.74	10.01	1.98 / 0.95	95.12 / 0.95
ami49-a	5154.64 / 0.75	15.82 / 0.88	1.54 / 0.88	98.32 / 0.98	6892.33 / 1	18 / 1	1.75 / 1	100 / 1	5304.4 / 0.77	8.96	0.86 / 0.49	100 / 1
ami49-b	11651.36 / 1.04	23.73 / 0.91	1.64 / 1.15	98.12 / 0.98	11252 / 1	26 / 1	1.42 / 1	100 / 1	8189.4 / 0.73	10.14	0.95 / 0.67	100 / 1
ami49-c	16164.18 / 1.22	30.09 / 0.67	1.66 / 1.06	97.66 / 0.98	13253 / 1	45 / 1	1.57 / 1	100 / 1	12764.6 / 0.96	12.84	1.3 / 0.83	100 / 1
ami49-d	19439.55 / 0.99	34.91 / 1.16	1.92 / 1.04	99.12 / 0.99	19632 / 1	30 / 1	1.85 / 1	100 / 1	16955.8 / 0.86	16.96	1.56 / 0.84	100 / 1
ami49-e	22277.9 / 0.99	48.1 / 0.84	2.42 / 0.79	99.8 / 1	22408 / 1	57 / 1	3.08 / 1	100 / 1	22452.6 / 1	21.86	1.5 / 0.49	100 / 1
Average	1.01	1.23	0.95	0.99	1	1	1	1	1.29	****	0.91	0.99

* The buses are connected to the block boundaries in both [5] and our approach.

** Successful Rate refers to the percentage of the floorplans generated by the SA in which all the buses are feasible.

*** The values after the slashes are the normalized values.

****[5] run on different machines, so we do not compare running time.

that of [5]. Similar to chapter 3, we just minimize the bus length and total area in our floorplanner because the works in [5] minimize bus length and total area without the fixed-outline constraint and the interconnect cost minimization. The buses are connected to the block boundaries in the three approaches (Table 4.1). The values after the slashes are the normalized values. The comparisons for all the data sets are reported in terms of bus length, running time, deadspace and successful rate. Successful rate refers to the percentage of the floorplans generated

by the SA have all the buses feasible. In [5], no results are reported for benchmark ami33-1 to ami33-5 and ami49-1 to ami49-6, so we do not compare those results. Compared with [5], their bus length is 1.29X of ours, their deadspace is 0.91X of ours and their successful rate is 0.99X of ours on average. We have also tried to tune the parameter of the SA process in order to produce similar dead space as in [5] and compare the bus length. The results are shown in Table 4.2. The approach in [5] is run on different machine, so we do not compare running time. Compared with our first floorplanner, the bus length is 1.01X of our second one, the running time is 1.23X of our second one, the deadspace is 1.2X of our second one and the successful rate is 0.99X of our second one on average.

In the second experiment, we handle the fixed-outline constraint and minimize both the bus length and the interconnect cost. In our experiment, deadspace threshold r is set to 10% and the aspect ratio R of the chip is 1.0. Therefore the W and H are computed as $\sqrt{1.1 \times \sum A_i}$ where A_i is the area of block b_i. The aspect ratio bound of a soft block b_i is $[t_i - 0.8, t_i + 0.8]$ where t_i is the original aspect ratio of block b_i. We use HPWL to estimate wire length. The information of the pins and net list is shown in Table 3.5. We compare our approach with our first floorplanner in chapter 3 and modified floorplanner of [4]. Experimental results are shown in Table 4.3. The results for all data sets are reported in terms of bus length, wire length, running time and successful rate. Experimental results show that the bus length of modified floorplanner of [4] is on average 2.33X of our bus length, their running time is on average 4.33X of our running time, their successful rate of our first floorplanner is on average 0.85X of ours, and their wire length is on average 1.04X of our wire length. Besides, experimental results also show that the bus length of our first floorplanner is on average 1.08X of our bus length, the running time of our first floorplanner is on average 2.29X of our running time, the successful rate of our first floorplanner is on average 0.98X of ours, and the wire length of

Table 4.2: Comparisons on Bus Length, Running Time, DeadSpace and Successfule Rate between Fixed-outline BDF with L-Shape Bus and [5] with Similar Deadspace

Test Cases	Fixed-outline BDF with L-shape bus				[5]			
	Bus Length*	Running Time (sec)	Deadspace (%)	Successful Rate **(%)	Bus Length*	Running Time (sec)	Deadspace (%)	Successful Rate (%)
ami33-a	477.5 / 1	5.43	0.98 / 1	100 / 1	710.5 / 1.49***	4.11	1.56 / 1.59	99.8 / 1
ami33-b	850.2 / 1	6.8	1.77 / 1	100 / 1	1366.7 / 1.61	6.13	1.67 / 0.94	99.28 / 0.99
ami33-c	1648.9 / 1	6.5	1.61 / 1	100 / 1	2433.2 / 1.48	7.35	1.72 / 1.07	98.18 / 0.98
ami33-d	1939.1 / 1	7.9	1.45 / 1	100 / 1	3318.4 / 1.71	8.05	1.87 / 1.29	97.81 / 0.98
ami33-e	2706.5 / 1	7.9	2.11 / 1	100 / 1	4578.6 / 1.69	8.91	1.88 / 0.89	95.46 / 0.95
ami33-f	2955.9 / 1	12	2.01 / 1	100 / 1	5078.6 / 1.72	10.01	1.98 / 0.99	95.12 / 0.95
ami49-a	6184 / 1	15	0.77 / 1	100 / 1	5304.4 / 0.86	8.96	0.86 / 1.12	100 / 1
ami49-b	11263 / 1	28	1.21 / 1	100 / 1	8189.4 / 0.73	10.14	0.95 / 0.79	100 / 1
ami49-c	15253 / 1	43	1.34 / 1	100 / 1	12764.6 / 0.84	12.84	1.3 / 0.97	100 / 1
ami49-d	20322 / 1	32	1.58 / 1	100 / 1	16955.8 / 0.83	16.96	1.56 / 0.99	100 / 1
ami49-e	23255 / 1	58	1.7 / 1	100 / 1	22452.6 / 0.97	21.86	1.5 / 0.88	100 / 1
Average	1	1	1	1	1.26	****	1.05	0.99

* The buses are connected to the block boundaries in both [5] and our approach.

** Successful Rate refers to the percentage of the floorplans generated by the SA in which all the buses are feasible.

*** The values after the slashes are the normalized values.

****[5] run on different machines, so we do not compare running time.

our first floorplanner is on average 1.03X of our wire length. The complexity and run time of our method do not increase with an increasing number of metal layers. A floorplanning solution is shown in Figure 4.7.

4.5 Summary

In this chapter, bus routability problem is handled. We present a floorplanner that can give a fixed-outline floorplanning solution including bus route that minimizes bus length, number of extra vias and interconnect cost. In each iteration of SA process, our goal is to route all bus components successfully by inserting an L-shape bus component for each bus component whose block alignment constraint is conflicting with that of other bus components. We first present a method to detect conflicting block alignment constraints and select some bus components to be con-

Table 4.3: Comparisons on Bus Length, Running Time, Wire Length and Successful Rate between Fixed-outline BDF, Fixed-outline BDF with L-shape Bus and Modified Floorplanner of [4]

Test Cases	Fixed-outline BDF**				Fixed-outline BDF with L-shape bus				Modified Floorplanner of [4]***			
	Bus Length*	Running Time (sec)	Wire Length	Successful Rate (%)	Bus Length*	Running Time (sec)	Wire Length	Successful Rate (%)	Bus Length*	Running Time (sec)	Wire Length	Successful Rate (%)
ami33-1	2593.5 / 1.09	15.12 / 2.2	88721.62 / 1.14	100 / 1	2385.88 / 1	6.88 / 1	78107.75 / 1	100 / 1	5295.2 / 2.22	24.8 / 3.6	89721 / 1.15	91.33 / 0.91
ami33-2	1132.14 / 1.61	22.86 / 2.86	87674 / 1.03	100 / 1	704 / 1	8 / 1	85202.86 / 1	100 / 1	4996 / 7.1	22.5 / 2.81	91004 / 1.07	88.32 / 0.88
ami33-3	1282.64 / 1.22	4.55 / 0.84	81766.91 / 1.03	98.35 / 0.98	1049.5 / 1	5.42 / 1	79005.58 / 1	100 / 1	1875.1 / 1.79	6.9 / 1.27	88012.9 / 1.11	74.47 / 0.74
ami33-4	3427 / 1.05	10.71 / 1.22	85935.29 / 1.03	99.01 / 0.99	3261.75 / 1	8.75 / 1	83199.25 / 1	100 / 1	5233.86 / 1.6	17.71 / 2.02	92861.29 / 1.12	87.73 / 0.88
ami33-5	5978.75 / 0.95	22.5 / 1.96	96991.88 / 1.04	100 / 1	6325 / 1	11.5 / 1	93162.5 / 1	100 / 1	8855 / 1.4	33 / 2.87	97217 / 1.04	80.06 / 0.8
ami33-a	1041.89 / 1.19	6.44 / 1.29	88626.33 / 1.09	96.83 / 0.97	875.6 / 1	5 / 1	81483 / 1	100 / 1	2151.89 / 2.46	8.56 / 1.71	84692.56 / 1.04	92.25 / 0.92
ami33-b	1435.33 / 1.07	9.11 / 1.56	90474.78 / 0.99	97.89 / 0.98	1339.83 / 1	5.83 / 1	91585.83 / 1	100 / 1	3823.8 / 2.85	11.7 / 2.01	86888.6 / 0.95	85.74 / 0.86
ami33-c	1704.44 / 0.86	12.56 / 1.98	93100.78 / 1.02	96.45 / 0.96	1976.83 / 1	6.33 / 1	91207.67 / 1	100 / 1	4223.4 / 2.14	16.7 / 2.64	92181.5 / 1.01	73.88 / 0.74
ami33-d	2446 / 1.12	13.75 / 2.29	95060.38 / 0.99	98.02 / 0.98	2180 / 1	6 / 1	96157 / 1	100 / 1	4864.71 / 2.23	21.43 / 3.57	95789 / 1	72.12 / 0.72
ami33-e	3276.44 / 0.9	16.56 / 1.91	99669.44 / 1.03	97.66 / 0.98	3648.33 / 1	8.67 / 1	96934 / 1	100 / 1	5875.86 / 1.61	28.14 / 3.25	95498.86 / 0.99	75.88 / 0.76
ami33-f	3291.91 / 0.91	20.91 / 2.09	76746.64 / 1.04	97.56 / 0.98	3630.12 / 1	10 / 1	73739.25 / 1	100 / 1	6473.86 / 1.78	23 / 2.3	91879.43 / 1.25	70.14 / 0.7
ami49-1	13395 / 1.05	33 / 2.83	1515660 / 0.98	100 / 1	12749.33 / 1	11.67 / 1	1544481.5 / 1	100 / 1	34523.5 / 2.71	79.67 / 6.83	1452900.67 / 0.94	91.01 / 0.91
ami49-2	14060.67 / 0.9	54.67 / 4.21	1602456 / 0.95	96.33 / 0.96	15667.83 / 1	13 / 1	1688731.67 / 1	100 / 1	38163.4 / 2.44	97.4 / 7.49	1430490.2 / 0.85	89.74 / 0.9
ami49-3	18584.83 / 0.97	58.67 / 4.51	1622330.67 / 1.06	98.92 / 0.99	19072 / 1	13 / 1	1526998.67 / 1	100 / 1	43292.33 / 2.27	191 / 14.69	1324948.33 / 0.87	85.28 / 0.85
ami49-4	12044.78 / 1.3	17.78 / 1.98	1207856 / 1.08	96.91 / 0.97	9249 / 1	9 / 1	1122439 / 1	100 / 1	12633.5 / 1.37	32.83 / 3.65	1204466.5 / 1.07	92.44 / 0.92
ami49-5	22143.33 / 1.26	34.56 / 2.38	1163613.11 / 0.98	97.01 / 0.97	17510.5 / 1	14.5 / 1	1183525 / 1	100 / 1	28692 / 1.64	72.5 / 5	1228114 / 1.04	89.25 / 0.89
ami49-6	22879.18 / 1.1	42.45 / 3.27	1102990.64 / 0.94	100 / 1	20864 / 1	13 / 1	1170611 / 1	100 / 1	38318 / 1.84	97.5 / 7.5	1303445.5 / 1.11	84.46 / 0.84
ami49-a	4511.33 / 1.01	16.67 / 2.08	1086896 / 1.01	97.34 / 0.97	4453 / 1	8 / 1	1078836 / 1	100 / 1	10066.3 / 2.26	35.4 / 4.43	1240945.6 / 1.15	92.24 / 0.92
ami49-b	9609 / 1.03	41.33 / 5.31	1150908.33 / 1.05	99.88 / 1	9332 / 1	7.78 / 1	1100516 / 1	100 / 1	16498.5 / 1.77	36.25 / 4.66	1285767 / 1.17	90.54 / 0.91
ami49-c	18672.2 / 1.16	22 / 1.57	1381215 / 1.04	97.9 / 0.98	16095 / 1	14 / 1	1323847 / 1	100 / 1	19302 / 1.2	61 / 4.36	1103986 / 0.83	83.76 / 0.84
ami49-d	19888 / 0.93	20 / 0.77	1459632 / 1.04	98.44 / 0.98	21413 / 1	26 / 1	1407030 / 1	100 / 1	NA /	NA /	NA /	NA /
ami49-e	27090 / 1.15	30 / 1.25	1396458 / 1.11	98.67 / 0.99	23615 / 1	32 / 1	1253930 / 1	100 / 1	NA /	NA /	NA /	NA /
Average	1.08	2.29	1.03	0.98	1	1	1	1	2.23	4.33	1.04	0.85

* The buses are connected to the block boundaries in both our floorplanner and the modified floorplanner of [4]. ** Fixed-outline BDF refers to the floorplanner described in the previous chapter. *** This modified floorplanner of [4] cannot give a feasible solution after running ten times on the data sets ami49-d and ami49-e.

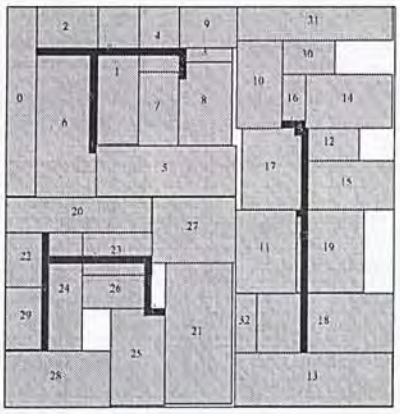

Figure 4.7: A bus routing solution $B_1 = \{0,1,2,3,4,5,6,7,8,9\}$, $B_2 = \{10,11,12,13,14,15,16,17,18,19\}$, $B_3 = \{20,21,22,23,24,25,26,27,28,29\}$

verted into L-shape bus components to make sure that there is no conflicting block alignment constraint. The selected bus components are then divided into L-shape bus components according to the method described in Section 4.3.1. In addition, some bus components satisfying certain conditions are combined to reduce the number of bus components. We list the conditions and also propose a method to combine bus components to reduce the number of bus components. Experimental results show that we also improve over the recent work [5] in terms of bus length and successful rate on average. Besides, we can improve over our first floorplanner and the modified floorplanner of [4] in terms of both bus length, running time and successful rate.

□ End of chapter.

Chapter 5

Conclusion

In this thesis, the fixed-outline BDF is discussed. We solve the fixed-outline BDF problem with an objective to minimize the total bus length and interconnect cost. In the first approach, we focus on the bus length minimization problem. Instead of adding up all the bus length in the cost function of the SA process, we handle the bus length minimization problem globally in a more sophisticated way by using a bus-driven partitioning step. In this step, min-cut partitioning is performed recursively to reduce bus length. At the same time, we use a more accurate estimation algorithm that connects buses to the block boundaries to estimate bus length. We can achieve shorter bus length on average by using this approach. Experimental results show that we can improve over the most closely related previous work [4] in terms of bus length, running time and dead space. Besides, we also improve over a most recent work [5] in terms of bus length on average.

Based on our first approach, we can increase the searching space to find a solution with more flexible bus shape and more flexible positions of the bus bendings. In the second approach, we focus on the bus routability problem. We handle the bus routability problem in a more sophisticated manner to increase the searching space and give more flexibility by changing the bus shape and the positions of the

bus bendings. We present a floorplanner that can give a fixed-outline floorplanning solution including bus route that minimizes bus length, number of extra vias and interconnect cost. Our goal is to route all the bus components successfully in each iteration of the SA process of this floorplanner. L-shape bus component is used to route the buses with conflicts with other bus components. Experimental results show that we also improve over the most recent work [5] in terms of bus length and successful rate on average. Besides, we can improve over our first floorplanner in Chapter 3 and the modified floorplanner of [4] in terms of both bus length, running time and successful rate. Future work will be focused on improving the run time for each test case.

□ End of chapter.

Bibliography

[1] Hua Xiang,Xiaoping Tang,Wong, M.D.F., bus driven floorplanning, *IEEE Trans.Computer-Aided Design of Integrated Circuits and Systems*,Vol.23, pages 1522- 1530, 2004.

[2] T.-C.Chen and Y.-W. Chang, Modern floorplanning based on B*-Tree and fast simulated annealing, *IEEE Trans. on Computer-aided design of integrated circuits and systems* ,vol. 25, No. 4, pages 637-650, 2006.

[3] Jill H.Y. Law, Evangeline F.Y. Young, Multi-bend bus driven floorplanning, *IEEE Trans.Integration, the VLSI Journal* , Vol. 41(2),pages 306-316, 2008.

[4] Tilen Ma, Evangeline F.Y. Young, TCG-based Bus Driven Floorplanning, *Proc. IEEE Asia South Pacific Design Automation Conference*,2008.

[5] B.-S. Wu and T.-Y. Ho, Bus-Pin-Aware Bus-Driven Floorplanning, *Proceedings of ACM Great Lake Symposium on VLSI* ,2010.

[6] G. E. Moore, Gramming More Components onto Integrated Circuits, *Electronics*, Vol.38, 1965.

[7] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar, Multilevel Hypergraph Partitioning: Applications in VLSI Domain, *IEEE Trans.on VLSI Systems*, Vol. 7, No. 1, pages 69-79, 1999.

[8] Tung-Chieh, Chen Yao-Wen, Chang Shyh-Chang Lin, IMF: interconnect-driven multilevel floorplanning for large-scale building-module designs, *Proc. 2005 IEEE/ACM International conference on Computer-aided design*, 2005.

[9] Saurabh N. Adya, Igor L. Markov, Fixed-outline Floorplanning: Enabling hierarchical design, *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*,vol. 11,no. 6, pages 1120-1135, Dec. 2003.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] Naveed Sherwani, Algorithm For VLSI Physical Design Automation, 3rd ed., Kluwer Academic Publishers, pages 193-196, 1999.

[26] G. Vijayan and R. Tsay, A new method for floorplanning using topological constraint reduction, *IEEE Trans. on Computer-Aided Design*, pages 1494-1501, Dec.,1991.

[27] B. Lokanathan and E. Kinnen, performance optimized floorplanning by graph planarization, *Proceedings of ACM/IEEE Design Automation Conference*, pages 116-121, 1989.

[28] W. W. Dai, B. Eschermann, E. Kuh, and M. Perdram, Hierarchical placement and floorplanning in bear, *IEEE Transactions on Computer-Aided Design*, Vol.8, pages 1335-1349, Dec. 1989.

[29] S. M. Sait, H.Youssef and M. S. T. Benten, Timing influenced general-cell genetic floorplanner, *Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95., IFIP International Conference on Hardware Description Languages; IFIP International Conference on Very Large Scale Integration., Asian and South Pacific*, pages 135-140, 1995.

[30] H. Youssef, S. M. Sait and K.J. Al-Farra, Timing influenced force directed floorplanning *Proceedings EURO-DAC'95*, pages 156-161, 1995.

[31] S. M. Sait and H.Youssef, VLSI Physical Design Automation - Theory and Practice, McGraw-Hill Book Company Europe and IEEE PRESS, pages 80-130, 1995.

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

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.