考慮匯流排腳位效應之匯流排導向平面佈局

摘要

由於在嵌入式系統中，匯流排的數量急劇地上升，匯流排規劃的品質好壞成為決定嵌入式系統效能與系統功率消耗的重要指標。為了不讓匯流排的問題造成晶片設計流程後期的限制，在早期平面佈局時就將匯流排的因素加入考慮是比較理想的處理方式。近年來，匯流排導向平面佈局的問題吸引了許多人的注意，並且已有相當多的方法被提出於處理相關的問題。然而，目前的演算法都採取了比較簡單的問題定義，忽略了匯流排腳位的位置和方向等相關的資訊。忽略了上述的資訊可能會造成系統效能被高估的情況。因此，我們提出了一個匯流排導向的平面佈局演算法，我們所提出的演算法在規劃匯流排時也考慮到腳位的位置與方向的資訊，讓演算法所得到的結果能夠更符合實際情況。由於在規劃匯流排的同時也考慮匯流排腳位的資訊，匯流排轉彎的地方並不局限在匯流排所連接的方塊上。因此，在處理匯流排

繞線問題時也比較有彈性。隨著整個匯流排拓撲的形狀變得更有彈性，匯流排繞線的成功率也相對地提高，許多較佳的結果就能被發掘出來。將我們的演算法所得到的結果與目前最佳的匯流排導向平面佈局演算法所得到的結果比較，我們演算法的執行時間相對快了3.5倍，繞線成功率提高了1.2倍，匯流排長度少了1.8倍，並且將平面佈局中空白的區域降低了1.2倍。

● 關鍵字:平面規劃;匯流排規劃

Bus-Pin-Aware Bus-Driven Floorplanning

Student: Po-Hsun Wu   Advisor: Dr. Tsung-Yi Ho

Department of Computer Science and
Information Engineering
National Cheng Kung University
Tainan, Taiwan, R.O.C.

Abstract

As the number of buses increase substantially in multi-core SoC designs, the bus planning problem has become the dominant factor in determining the performance and power consumption of SoC designs. To cope with the bus planning problem, it is desirable to consider this issue in early floorplanning stage. Recently, bus-driven floorplanning problem has attracted much attention in the literature. However, current algorithms adopt an over-simplified formulation ignoring the position and orientation of the bus pins, the chip performance may be deteriorated. In this paper, we propose the bus-driven floorplanning algorithm that fully considers the impacts of the bus pins. By fully utilizing the position and orientation of the bus pins, bus bendings are not restricted to occur at the modules on the bus, then it has more flexibility during bus routing. With more flexibility on the bus shape, the size of the solution space is increased and a better bus-driven floorplanning solution can be obtained. Compared with the bus-driven floorplanner [6], the experimental results show that our algorithm performs better in runtime by 3.5×, success rate by 1.2×, wirelength by 1.8×, and reduced the deadspace by 1.2×.

- **Keywords:** Floorplanning; Bus planning

Table of Contents

List of Tables **vii**

List of Figures **viii**

Chapter 1. Introduction **1**

1.1 Previous Works 5

1.2 Our Contributions 6

Chapter 2. Problem Formulation **9**

Chapter 3. Constraints and Terminologies **11**

Chapter 4. Algorithm **16**

4.1 Modified Prim's Algorithm 18

4.2 Segment Creation 19

4.3 Bus Ordering and Coordinate Determination 20

4.3.1 Bus Ordering 20

4.3.2 Coordinate Determination 22

4.4 Wirelength Reduction 23

4.4.1 Overlap Minimization 23

4.4.2 Coordinate Adjustment 25

4.5 Layer Assignment 29

4.6 Orientation Determination and Deviation Minimization 31

4.6.1 Deviation Minimization with Lookup Table Method 31

4.6.2 Fast Deviation Update Based on Topology Comparison . . 35

4.7 Soft Module Adjustment 42

Chapter 5. Experimental results **43**

5.1 Experiments on Regular Testcases for [6] and [11] 43

5.2 The Comparison Between [11] and This Work 46

5.3 The Comparison Between Modified [11] and This Work 47

5.4 Experiments on Larger Testcases for [11] and This Work 48

Chapter 6. Conclusions **51**

Chapter 7. Acknowledge **52**

Bibliography **53**

List of Tables

List of Figures

1.1 (a) One horizontal bus connects two bus pins. (b) Bus routing without considering the effect of the bus pin. (c) Bus twisting. (d) No bus twisting after flipping the bus pin. 2

1.2 (a) Simplified bus routing. (b) Bus routing considering the bus pin. (c) Bus routing with the diagonal connection between the modules. (d) Via-aware bus routing with the diagonal connection between modules. (e) Bus routing considering both vias and wirelength deviation. 3

3.1 (a) Capacity constraint. (b) A routable solution under the capacity constraint. 11

3.2 Vertical overlapping. 12

3.3 Bus pin flipping. 13

3.4 The possible orientation directions. 14

4.1 Algorithm overview. 16

4.2 (a) Minimum spanning tree. (b) The resulting routing tree. . 18

4.3 (a) Before adding segment. (b) After adding segment. 19

4.4 (a) Bus ordering. (b) Bus crossing. (c) Ordering constraint graph. 20

4.5 (a) Module moving. (b) Bus overlapping. (c) No bus overlapping. 22

4.6 (a) Before wirelength reduction. (b) After wirelength reduction. . 23

4.7 (a) Reduction rule for horizontal bus components. (b) Reduction rule for vertical bus components. 24

4.8 (a) Wirelength reduction rule. (b) Before wirelength reduction. (c) After wirelength reduction. 25

4.9 The original bus topology. 27

4.10 The rules for wirelength reduction. 28

4.11 Layer assignment. 30

4.12 Total possible bus shapes between any two modules. 31

4.13 The bus patterns derived from the horizontal, vertical, and diagonal connections. 32

4.14 Determine the orientation and deviation for all bus pins and turning nodes. 33

4.15 The reduced bus patterns. 35

4.16 The procedure of deviation determination and AD update. . 36

4.17 Determine the orientation and deviation for all bus pins and turning nodes. 37

4.18 The difference between two algorithms. 38

4.19 All possible topologies in driver. 39

4.20 All possible topologies in load. 41

5.1 (a) Packing result of ami33-i. The buses are {13, 16, 18, 19, 21, 27, 30}, {0, 5, 6, 20, 25, 30, 32}, {14, 15, 16, 17, 24, 28, 32}. (b) Packing result of ami49-h. The buses are {7, 8, 9, 10, 11, 12, 41}, {29, 32, 41, 43, 44, 46, 47}, {0, 1, 2, 3, 4, 9, 46}. . . 46

5.2 (a) Packing result of n100-c. The buses are {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 5, 15, 25, 35}, {1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 6, 16, 26, 36}, {2, 12, 22, 32, 42, 52, 62, 72, 82, 92, 7, 17, 27, 37}, {3, 13, 23, 33, 43, 53, 63, 73, 83, 93, 8, 18, 28, 38}, {4, 14, 24, 34, 44, 54, 64, 74, 84, 94, 9, 19, 29, 39}. (b) Packing result of n100-f. The buses are {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 5, 15, 25, 35, 45, 55, 65, 75, 85, 95}, {1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 6, 16, 26, 36, 46, 56, 66, 76, 86, 96}, {2, 12, 22, 32, 42, 52, 62, 72, 82, 92, 7, 17, 27, 37, 47, 57, 67, 77, 87, 97}, {3, 13, 23, 33, 43, 53, 63, 73, 83, 93, 8, 18, 28, 38, 48, 58, 68, 78, 88, 98}, {4, 14, 24, 34, 44, 54, 64, 74, 84, 94, 9, 19, 29, 39, 49, 59, 69, 79, 89, 99}. 50

Chapter 1

Introduction

With the increasing design complexity, the amount of buses between different modules on a chip also increases rapidly. Bus routing has become a major concern of comparable importance to timing and power in multi-core SoC designs. To ease the efforts of bus routing in later routing stage, it is desirable to consider this issue in early floorplanning stage. Since buses are of different widths and required to go through different sets of modules, the bus routability is severely affected by the module position. Bus-driven floorplanning targets on obtaining a bus-routable floorplan such that the chip area and the bus area are minimized. Recently, many bus-driven floorplanning algorithms have been proposed in the literature [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. However, these bus-driven floorplanning algorithms adopt an over-simplified formulation which ignores the practical issues such as the position and orientation of the bus pin. Without taking those practical issues into consideration, it may have the following impacts on bus routing:

- **Bus twisting:** It makes the signal wires cross at a point and decreases accessibilities of the bused pins.
- **Via increasing:** Several vias occur at the bend of a bus that have

adverse effects on the bus delay.

- **Delay variation:** Different driver-load wirelength between different bus bits causes delay variation among all bus bits.

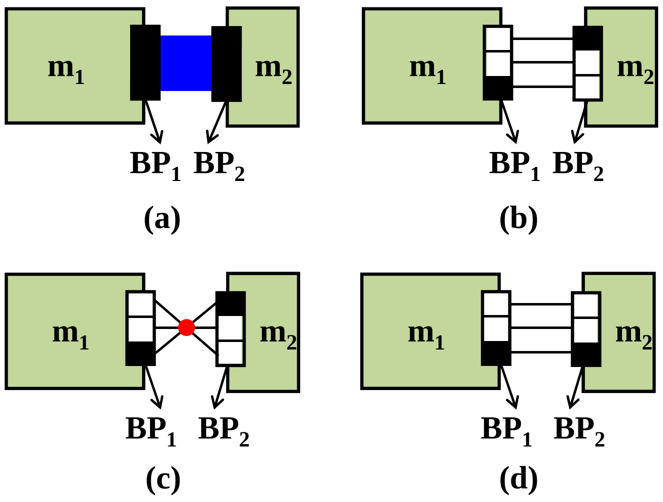

Figure 1.1: (a) One horizontal bus connects two bus pins. (b) Bus routing without considering the effect of the bus pin. (c) Bus twisting. (d) No bus twisting after flipping the bus pin.

One example of the bus twisting is given in Figure 1.1. In Figure 1.1 (a), two modules m_1 and m_2 are placed on the floorplan, and two bus pin BP_1 and BP_2 connected by one horizontal bus are placed on the boundary of the two modules. Since the effect of the bus pin is not considered during bus routing in current algorithms, the routing result in Figure 1.1 (b) is obtained, and some bits of the bus pin are not correctly connected by the wire. Therefore, the wrong signal is transmitted via the bus, the Most Significant Bit (MSB) of each bus pin is marked with black color in all figures of this paper. In this paper, the bus pin is carefully handled during bus routing, thus, each bus can be correctly connected by the wire as shown in Figure 1.1 (c). However, all wires are twisted at one point in the figure, the problem can be solved by flipping the bus pin BP_2 as shown in Figure 1.1 (d).

Therefore, an over-simplified bus-driven floorplanning formulation will make the routability and the solution quality in doubt.

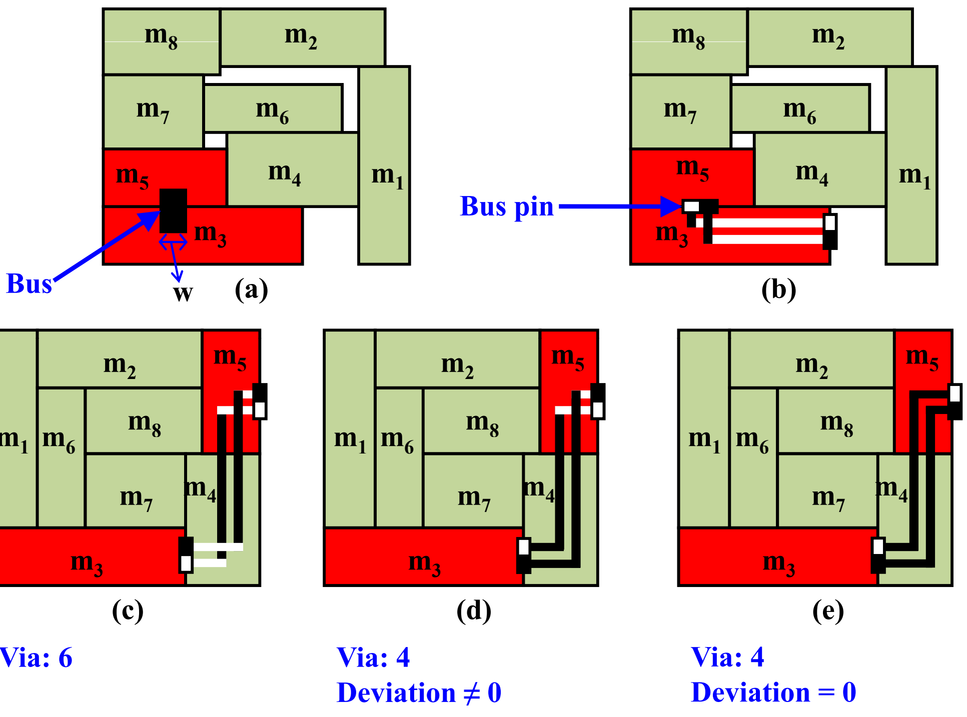

Figure 1.2: (a) Simplified bus routing. (b) Bus routing considering the bus pin. (c) Bus routing with the diagonal connection between the modules. (d) Via-aware bus routing with the diagonal connection between modules. (e) Bus routing considering both vias and wirelength deviation.

Figure 1.2 illustrates the impact of the position and orientation of the bus pin. There are eight modules and one bus that connects two modules m_3 and m_5, i.e., bus modules. In Figure 1.2 (a), existing bus-driven floorplanning algorithms connect m_3 and m_5 by simply one vertical bus. However, by considering the actual positions of the bus pins in Figure 1.2 (b), the bus pins of these two modules can not be passed by only one vertical bus, and the wirelength is much longer than the simplified estimation.

Furthermore, current bus-driven floorplanning algorithms restrict a bus only bends above the bus modules, i.e., only horizontal and vertical connections on the bus modules are allowed. Such restriction limits the flexibility on the bus shape resulting a narrower solution space. Therefore, better bus-driven floorplanning solutions may be missed in such formulation. In this paper, we explore the diagonal connection between the bus modules to improve the solution quality. Compared with Figure 1.2 (b), a better solution with smaller area can be achieved by utilizing the diagonal connection between m_3 and m_5 as shown in Figure 1.2 (c). Although there are two bends occur at two modules m_4 and m_5, no extra vias are required at the bend of m_5 since vias exist anyway to connect to the bus modules m_5. Even extra vias are still required at another bend on the module m_4 which is not a bus module, it can be eliminated by simply flipping the bus pin on the module m_3 and assigning the horizontal and vertical buses on the module m_4 to the same layer as shown in Figure 1.2 (d).

As mentioned earlier, different driver-load wirelength between bus bits causes delay variation among all bus bits. At a load, the wirelength difference among all bus bits is characterized by **wirelength deviation**. Different from Figure 1.2 (d), the driver-load bus delay among all bus bits is different, the deviation can be further optimized to 0 by flipping the bus pin on module m_5 as shown in Figure 1.2 (e).

In this paper, we propose a bus-driven floorplanning algorithm that fully considers the impact of the bus pins. By fully utilizing the position and orientation of the bus pins, we can obtain the high-quality results by exploring flexibility on the bus shape. Moreover, we also develop two effective algorithms to minimize the wirelength deviations and two wirelength reduction algorithms to optimize the total bus wirelength. Experimental results show that our algorithms are very promising.

1.1 Previous Works

The bus-driven floorplanning problem has recently attracted a lot of attention in the literature. Rafiq et al. proposed an integrated floorplanning for bus-based microprocessor designs [1]. Multiple topologies were produced for each bus, and a mixed integer/linear programming approach was adopted to arbitrate the conflicts between different buses. However, the algorithm was limited to single fanout buses. Xiang et al. [2] solved the bus-driven floorplanning problem by using a sequence pair (SP) representation based on a simulated annealing (SA) framework. Chen and Chang [3] handled the bus-driven floorplanning problem based on the B*-tree representation. The drawback of the above two methods is that all modules are connected by either horizontal or vertical 0-bend buses; the solution space is thus restricted when a large number of the modules are involved for each bus. The bus shape problem was improved by [4], where 0-bend, 1-bend, and 2-bend buses were allowed. Nevertheless, there are still limitations on the bus shape. In [6], a multi-bend bus-driven floorplanning algorithm was proposed based on a TCG representation. It has more flexibility on bus shapes and obtains higher success rate.

In microprocessors and DSPs, many cores are involved and the signals are transmitted between different cores by the buses which play an important role in determining the chip performance. Therefore, some previous works were developed to deal with such floorplanning in microarchitectural design. Kim and Lim [7] presented a new bus routing algorithm based on Hanan grids and linear programming. In [8], the authors proposed a bus-aware microarchitectural floorplanning which addressed the impact of bus routability on performance/power/thermal issues. However, the above two works aimed on optimizing IPC (Instructions Per Cycle) and had a different problem formulation from those aforementioned floorplanners.

There are still many works which consider other constraints during the bus planning. Xiang et al. [5] proposed an OPC-friendly bus driven floorplanning algorithm which carefully designed the pitch for buses to avoid the forbidden pitch. Sheng et al. [9] proposed an approach which was based on a deterministic algorithm Less Flexibility First, it run in a fixed-outline area and packed hard blocks one after another. In [10], they presented a floorplan revising method which was based on linear programming model to minimize the number of reducible routing vias with a controllable loss on the chip area and wirelength. The authors [11] considered the bus pin issue and explored the diagonal connection which makes the bus shape more flexible and improves the solution quality.

1.2 Our Contributions

In this paper, we propose a bus-driven floorplanning algorithm that fully considers the impact of the bus pins. The diagonal connection between different modules is explored to make the bus shape more flexible which improves the success rate. The contributions of this paper are listed as follows:

- **Deviation Minimization:** We develop two algorithms to minimize the wirelength deviation for signal integrity of the buses. First algorithm explores all possible topologies between any two modules, and it finally concludes some specific bus patterns, then the best deviation at each module can be quickly obtained from those specific bus patterns. In second algorithm, the procedure of determining the deviation at each module is divided into two steps — deviation determination and accumulated deviation update. During determining the deviation at one module, all the accumulated deviation from the driver to its neighbors can also be calculated. Since the procedure of determining the deviation at each module is divided into two steps, the segment size for each comparison becomes smaller, then the total bus patterns can be further reduced to 10 types. Fewer bus patterns stands for less time is needed for searching the possible bus patterns. Experimental results show that the wirelength deviation can be effectively reduced.

- **Diagonal relation:** In this paper, we explore the diagonal relation between different modules to make the bus shape more flexible, then the modified graph coloring algorithm is proposed to determine the layer of each bus. Experimental results show that our algorithm has 1.2× improvement on the success rate compared with [6].

- **Wirelength reduction:** To further enhance the solution quality, we present two wirelength reduction algorithms to improve the total bus wirelength. First algorithm reduces the total bus wirelength by minimizing the overlap between different bus components (for the multi-bend bus, each horizontal (vertical) part is called a bus component) during constructing the minimum spanning tree (MST). The second algorithm is used to optimize the bus wirelength by adjusting the coordinate of each horizontal (vertical) bus component.

The rest of this paper is organized as follows. Chapter 2 gives the problem formulation of the bus-driven floorplanning. Chapter 3 defines the constraints and terminologies used in this paper. Chapter 4 gives the details of the proposed bus-driven floorplanning algorithm. Experimental results are shown in Chapter 5. Finally, a conclusion will be given in Chapter 6.

Chapter 2

Problem Formulation

Since long or timing critical buses are normally assigned to a pair of high metal layers, one horizontal and one vertical, we only consider the two-layer bus routing problem. The horizontal and vertical buses can be routed in either layer. In the bus-pin-aware bus-driven floorplanning problem, we are given the following:

1. A set of n modules $M = \{m_1, m_2, ..., m_n\}$, each module m_i is associated with height h_i and width w_i, where $h_i, w_i \in R^+$.

2. A set of m buses $B = \{b_1, b_2, ..., b_m\}$, each bus b_j has a width t_j and goes through a set of modules. Moreover, the position of each bus pin is also defined in each bus, where $t_j \in R^+$.

Our goal is to determine the orientation and position of the bus pins on each module and decide the routing path for each bus such that no overlapping occurs between different horizontal (vertical) bus components. Moreover, no overlapping is allowed between different modules. The objectives of the bus-pin-aware bus-driven floorplanning problem are:

(1) to **minimize the chip area**,

(2) to **minimize the bus area**,

(3) to **minimize the number of infeasible buses**,

(4) to **minimize the deviation of each bus**.

Chapter 3

Constraints and Terminologies

In this section, the capacity constraint in bus-pin-aware bus-driven floorplanning are defined, followed by the introduction of the terminologies used in this paper.

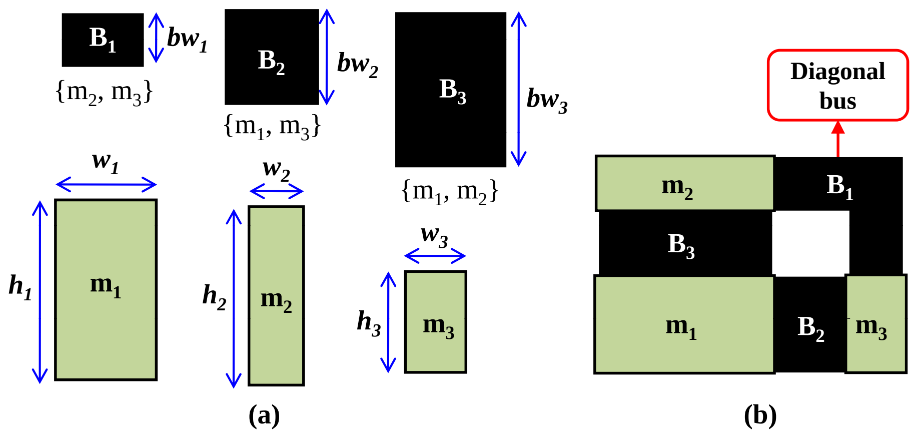

Figure 3.1: (a) Capacity constraint. (b) A routable solution under the capacity constraint.

Capacity constraint is illustrated in Figure 3.1 (a). There are three buses $B_1 = \{m_2, m_3\}$, $B_2 = \{m_1, m_3\}$, and $B_3 = \{m_1, m_2\}$, moreover, three modules m_1, m_2, and m_3 are placed on the floorplan. bw_i is the bus width of bus B_i, and the width and height of each module m_i are denoted by w_i and h_i, respectively.

Since the capacity of each module is limit, some modules can be passed with only one bus at the same direction. For instance, $bw_2 + bw_3 > \max(w_1, h_1)$, therefore, the capacity of the module m_1 is not enough for both buses B_2 and B_3 to pass through.

Compared with existing bus-driven floorplanning algorithms which result in an unroutable solution in this example, we can have a routable solution by exploring the diagonal connection between different modules in Figure 3.1 (b). The modified graph coloring algorithm is proposed to assign the bus components of the diagonal bus B_1 to the same layer such that no extra via is required at the bend of the bus B_1.

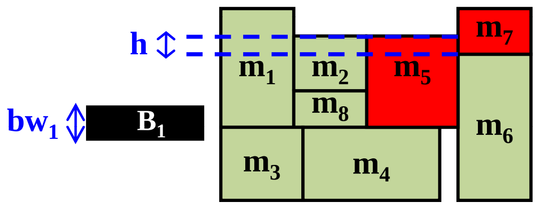

Figure 3.2: Vertical overlapping.

Another constraint is vertical overlapping as shown in Figure 3.2, the bus width of B_1 is bw_1 and it connects two modules m_5 and m_7, the overlapped range between two modules is h. If $h < bw_1$, then it has no sufficient space for the bus B_1 to pass through that the vertical overlapping constraint is violated, thus, the coordinate of the module m_5 must be adjusted to increase the overlapped range.

Terminologies used in this paper are introduced as follows:

Definition 1: A ***bus pin*** of an n-bit bus consists of n pins. The bits of each bus pin are arranged monotonically from the Least Significant Bit (LSB) to the MSB, and are equally spaced. The MSB bit is marked with black color in all figures of this paper. A bus pin is oriented horizontally or vertically. For example, the bus pins BP_1, BP_2, and BP_6 are oriented horizontally, and the bus pins BP_3, BP_4, and BP_5 are oriented vertically in Figure 3.3 (a).

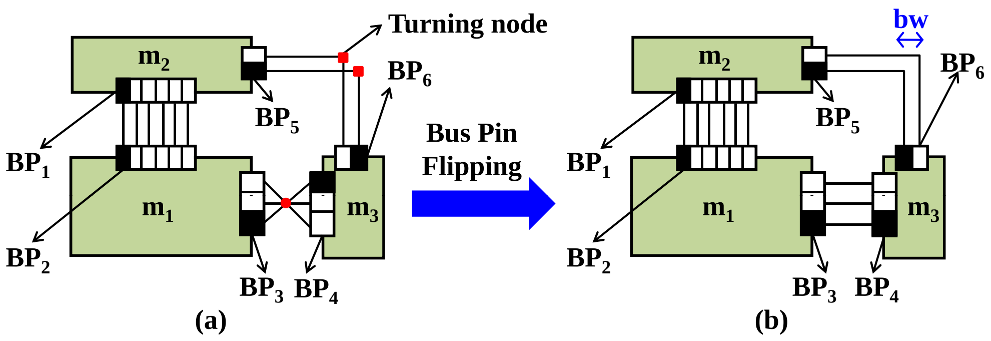

Figure 3.3: Bus pin flipping.

Definition 2: The ***position of the bus pin*** is the position of the bus pin on the module, it can be placed on one of the four boundaries of the modules. For example, the bus pin BP_1 is placed on the lower boundary of the module m_2 in Figure 3.3 (a).

Definition 3: ***Turning nodes*** denote the bends of the buses and they are connected with the horizontal and vertical buses. For example, there are two turning nodes at the bend of the bus connecting the bus pins BP_5 and BP_6 in the Figure 3.3 (a).

Definition 4: The ***orientation of the bus pin*** is defined as the direction from the LSB to the MSB. For example, the orientation of the bus pin BP_1 is from east to west, and the orientation of the bus pin BP_3 is from north to south in Figure 3.3(a). All possible orientations for the bus pins and the turning nodes are listed in Figure 3.4.

Definition 5: ***Bus pin flipping*** is used to change the orientation of the bus pins. In Figure 3.3 (a), the orientation of the bus pin BP_3 is from north to south and the orientation of the bus pin BP_4 is from south to north. Since

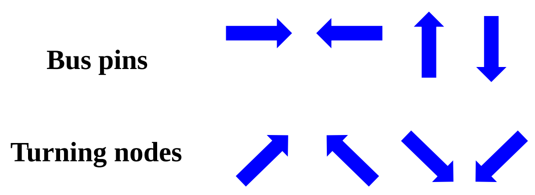

Figure 3.4: The possible orientation directions.

their orientation is opposite and they are connected with one horizontal bus, the bus twisting makes the signal wires cross at a point. Furthermore, the bus pin BP_5 and BP_6 are connected with a diagonal bus, the orientation of the bus pin BP_5 is from north to south and the orientation of the bus pin BP_6 is from west to east, two extra vias is required at the turning nodes as shown in Figure 3.3 (a). All the above problems can be solved by simply applying bus pin flipping on the bus pins BP_4 and BP_6. The result after bus flipping is shown in Figure 3.3 (b).

Definition 6: Wirelength "***deviation***" represents the driver-load wirelength difference among all bus bits. Different choices of the orientations at the bus pins and turning nodes may cause different driver-load wirelength among all bits. In this paper, we only need to minimize the wirelength difference between MSB and LSB of the bus because the wirelength of all other bits can be linearly interpolated by it. The MSB-LSB wirelength deviation could be expressed by

$$dev_{len} = |len(MSB) - len(LSB)| \tag{3.1}$$

A turning node can contribute −D, 0 or +D to the MSB-LSB wirelength difference, where D = 2×BW and BW is the bus width. Accumulating the turning node contributions along the path from the driver to the load of the bus provides dev_{len} at the load. Figure 3.3 illustrates the different deviation results from the different choices at the turning nodes. In Figure 3.3 (b), the deviation from the bus pin BP_5 to the bus pin BP_6 is $-$D, while in Figure 3.3 (a) is 0, but it needs two extra vias at the turning nodes. Therefore, it is hard to optimize the total deviation with no extra vias are used at the turning nodes.

Chapter 4

Algorithm

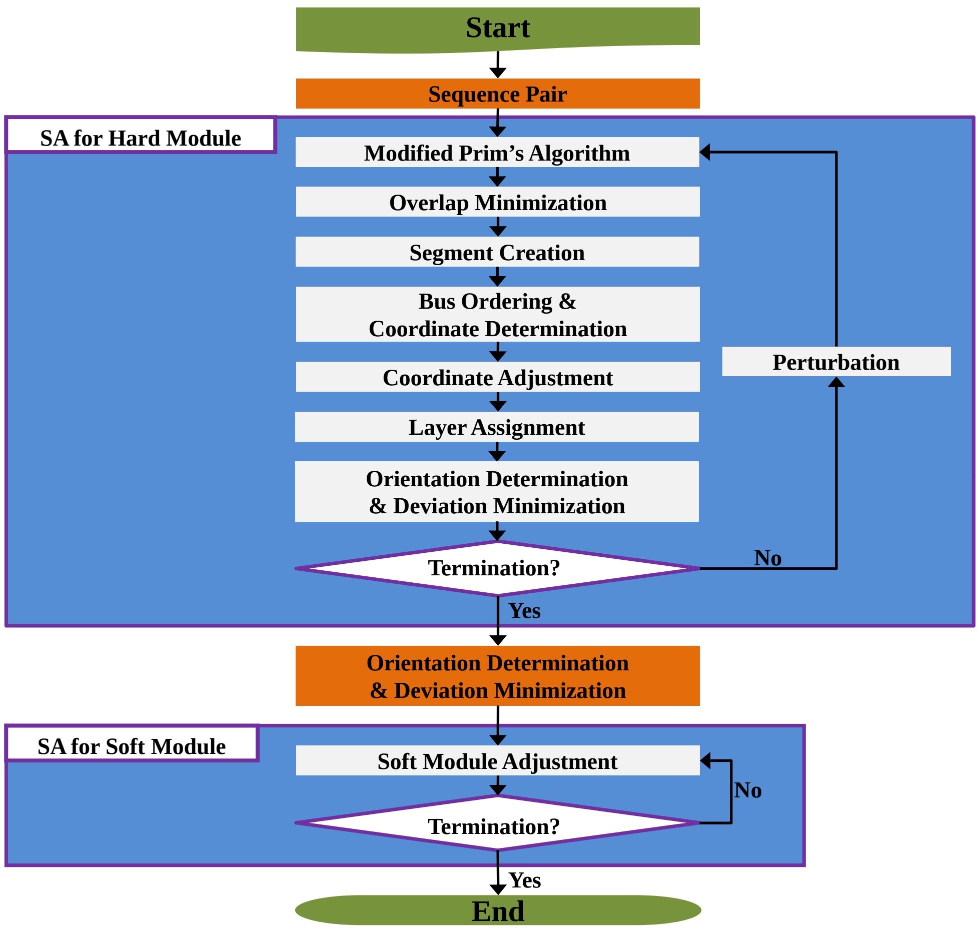

Figure 4.1: Algorithm overview.

The flowchart of our bus-pin-aware bus-driven floorplanning algorithm is shown in Figure 4.1. In the beginning of our algorithm, it derives a floorplan by using the sequence pair representation [14], then it computes the coordinate of each module on the floorplan to obtain the geometric relations between modules. Next, the modified Prim's algorithm is used to obtain bus routing topologies, and it applies the overlap minimization algorithm to minimize the overlap between different bus components in the MST. Since some bus pins may not be passed by the buses, it needs to add some buses for connecting the bus pins to the MST. After the above steps, several horizontal and vertical buses are obtained, and their coordinate could be determined. Then, it performs coordinate adjustment algorithm to further optimize the wirelength by adjusting the coordinate of each bus. Since extra vias are ***forbidden*** at the bend of the diagonal buses, the modified graph coloring algorithm is used to assign each bus to different layers. Then the algorithm discussed in the Section 4.6.2 is used to determine the position and orientation of each bus pin for the current solution, meanwhile, it minimizes the deviation at each load.

During perturbation, it has three operations to perturb the current floorplan: (1) Rotate. (2) Reverse. (3) Swap. The cost function used in this paper is defined as follows:

$$Cost = \alpha \cdot \boldsymbol{A} + \beta \cdot \boldsymbol{B} + \gamma \cdot \boldsymbol{I} \tag{4.1}$$

where $\boldsymbol{A}$ is the chip area, $\boldsymbol{B}$ is the bus area, $\boldsymbol{I}$ is the number of invalid bus nets, α, β, and γ are parameters defined by the users. For saving runtime, it only performs the bus routing algorithm when the current chip area is smaller than the chip area of the best solution. Each time a better solution is found, it will check the deviation of the best solution and the current solution, the solution with best deviation will be stored.

After the simulated annealing stage, the algorithm discussed in the Section 4.6.2 is performed to determine the position and orientation of each bus pin for the best solution. Finally, it changes either the width or height of the modules lying on the critical path to obtain a better chip area.

4.1 Modified Prim's Algorithm

To derive the bus topologies, it first constructs the MST for connecting the bus modules. Since the deviation is calculated from the driver to each load, to take the impact of the deviation into consideration, we adopt the Prim's algorithm to construct the MST from the driver. During constructing the MST, it checks the capacity of each module to avoid violating capacity constraint. If the selected edges violate the constraint, then other edges will be chosen to connect the MST. The bus is regarded as infeasible if no MST is finally constructed.

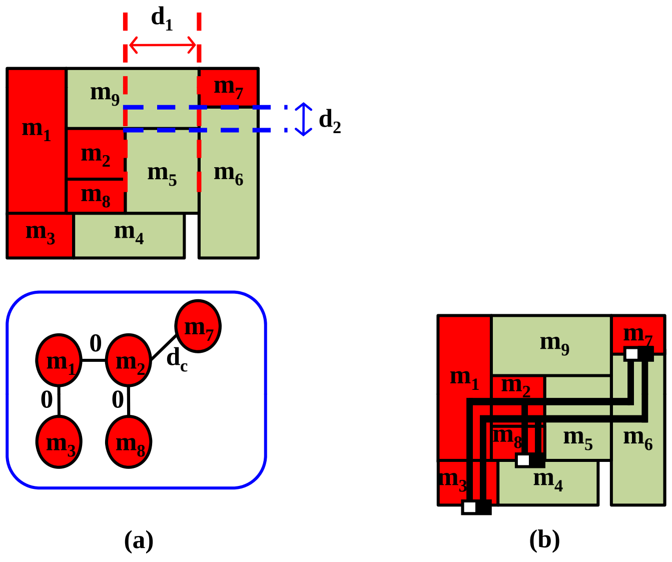

Figure 4.2: (a) Minimum spanning tree. (b) The resulting routing tree.

Figure 4.2 (a) is an example of the MST. The MST connects the modules

m_1, m_2, m_3, m_7, and m_8, the weight of each edge is derived from the distance between different modules. For instance, the weight d_c of the diagonal bus connecting m_2 and m_7 is $d_1 + d_2$. When determining the coordinate of each bus, it only handles the horizontal and vertical buses, thus, each diagonal bus in the MST will be split into one horizontal and one vertical buses. Figure 4.2 (b) shows the routing result of Figure 4.2 (a).

During constructing the MST, it considers the diagonal connection between different modules, checks the capacity constraint, and minimizes the bus wire-length. Therefore, we modify the Prim's algorithm to meet the above requirements.

4.2 Segment Creation

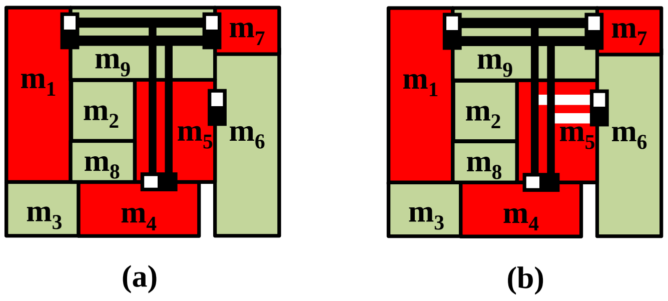

Figure 4.3: (a) Before adding segment. (b) After adding segment.

During constructing the MST, the position of the bus pins is ignored, as a result, the bus pin of some modules may not be passed by the MST. To connect the bus pin to the MST, some segments must be added to meet the requirement. Figure 4.3 (a) is an example, the bus connects four modules m_1, m_4, m_5, and m_7, and the bus pin on module m_5 is not passed by the MST. To solve this problem, one horizontal segment is added for connecting the bus pin on the module m_5 to the MST. The result of segment addition is shown in Figure 4.3 (b).

4.3 Bus Ordering and Coordinate Determination

4.3.1 Bus Ordering

Since the coordinate of each bus is determined in non-decreasing order, it finds the bus ordering of all horizontal (vertical) buses and determines the coordinate of each bus according to the bus ordering [2]. Given a sequence pair, it can obtain the relative position of any two modules, and the ordering of any two buses is derived from the relative position of those modules passed by the buses.

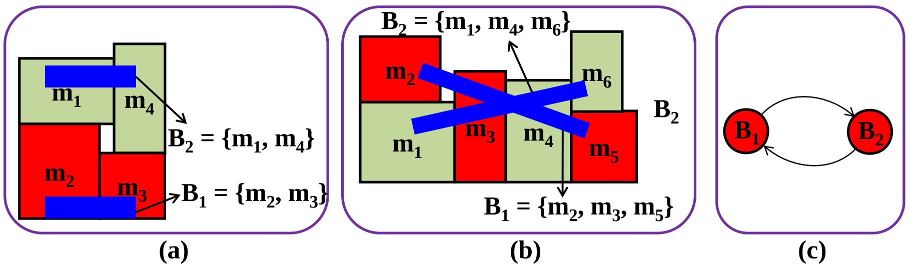

Figure 4.4: (a) Bus ordering. (b) Bus crossing. (c) Ordering constraint graph.

Figure 4.4 (a) gives an example. The sequence pair of the floorplan is (1243, 2314), two buses $B_1 = \{m_2, m_3\}$ and $B_2 = \{m_1, m_4\}$ are placed on the floorplan. The module m_1 is placed above the module m_2 according to the sequence pair, thus, the bus B_2 passing through the module m_1 has to be placed above the bus B_1 passing through the module m_2. This condition is called *bus ordering*. In order to obtain the bus ordering between any two buses, it constructs the ordering constraint graph (OCG) by performing the above steps on any two horizontal (vertical) buses. The buses are represented as the nodes in the OCG, and the edges represent the ordering between any two buses.

During determining the bus ordering, different buses may conflict with each other. In Figure 4.4 (b), two buses $B_1 = \{m_2, m_3, m_5\}$ and $B_2 = \{m_1, m_4, m_6\}$ are placed on the floorplan. Based on the sequence pair, the module m_2 is placed above the module m_1, and the module m_6 is placed above the module m_5. Since the modules m_1 and m_2 are passed by the buses B_2 and B_1, respectively, an edge from the node B_1 to the node B_2 is inserted into the OCG. Besides, the modules m_5 and m_6 are passed by the buses B_1 and B_2, respectively, an edge from node B_2 to node B_1 is inserted into the OCG. Thus, the OCG contains a cycle as shown in Figure 4.4 (c). It means that two buses conflict with each other, and one of the two buses is regarded as infeasible. This situation is called *bus crossing*.

To determine the bus ordering of all horizontal (vertical) buses from the OCG, we first set the nodes with zero out-degree in the OCG the highest priority comparing with remaining horizontal (vertical) buses. By this property, it determines the bus ordering by deleting the nodes and edges from the OCG iteratively. In each iteration, it removes the node with zero out-degree and deletes the edges connecting to it from the OCG. If no such node in the OCG, it means that the cycle is contained in the OCG and one of the minimum out-degree nodes is regarded as infeasible. Then the invalid node and the edges connecting to it will be deleted from the OCG. The above steps are repeated until all nodes are removed from the OCG. Finally, the bus ordering of all feasible horizontal (vertical) buses can be obtained.

4.3.2 Coordinate Determination

After determining the bus ordering, it starts to determine the coordinate of all horizontal (vertical) buses [2]. Without loss of generality, we take the horizontal buses for example. The coordinate of each horizontal bus B_i is $y_{max} = \max\{y_i \mid i = 1, 2, ..., \text{k}\}$, where k is the number of the modules passed by the bus, and y_i is the y coordinate of each module. If some modules are not passed by the bus B_i. It adjusts the coordinate of each module slightly such that all modules can be passed by the bus B_i.

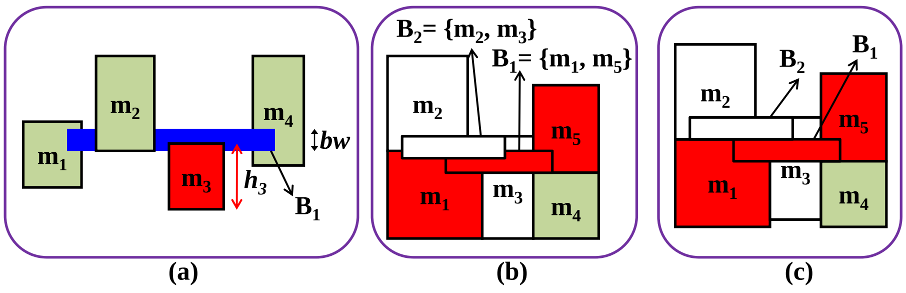

Figure 4.5: (a) Module moving. (b) Bus overlapping. (c) No bus overlapping.

In Figure 4.5 (a), the bus B_1 goes through the modules in $\{m_1, m_2, m_3, m_4\}$, the bus width of B_1 is bw, the height of the module m_i is h_i. In this figure, the coordinate of B_1 is $y_{max} = \max\{y_i \mid i = 1, 2, 3, 4\}$. Since the module m_3 is not passed by the bus B_1, it changes the coordinate of the module m_3 slightly such that it can be passed by B_1, the new coordinate of m_3 is y_{max} + bw - h_3.

When considering multiple horizontal (vertical) buses, different horizontal (vertical) buses may overlap with each other. In Figure 4.5 (b), the buses $B_1 = \{m_1, m_5\}$ and $B_2 = \{m_2, m_3\}$ are placed on the floorplan. The bus B_2 overlaps with the bus B_1 on the floorplan, this situation is called *bus overlapping*.

Therefore, the bus B_2 has to be moved up until there is no overlap with the buses below. The result is shown in Figure 4.5 (c).

4.4 Wirelength Reduction

4.4.1 Overlap Minimization

Since the width and height of each module is not considered during constructing the MST, the bus wirelength may be further optimized. In this paper, we propose an algorithm to reduce the overlap in MST such that the better bus wirelength could be obtained.

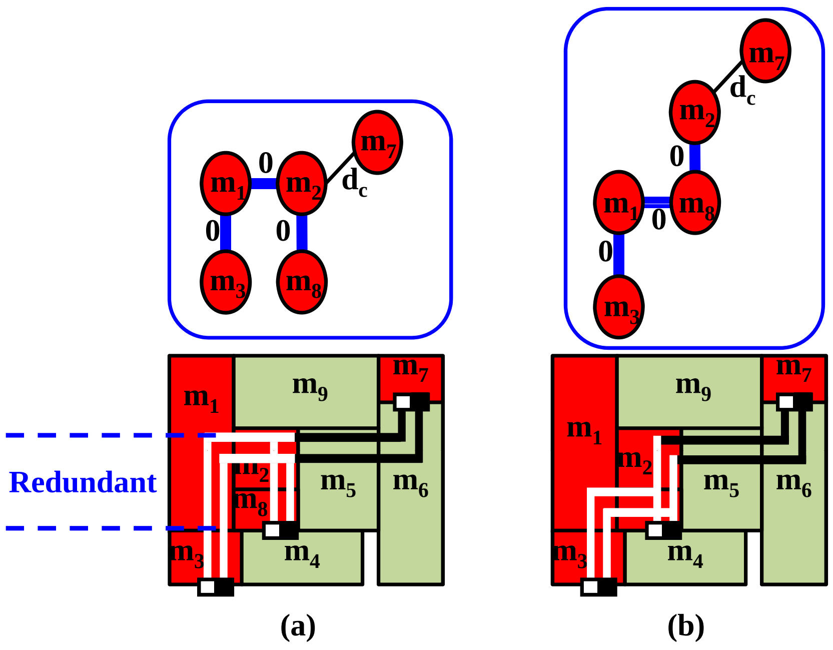

Figure 4.6: (a) Before wirelength reduction. (b) After wirelength reduction.

Figure 4.6 illustrates how the algorithm works. The bus connects the five modules m_1, m_2, m_3, m_7, and m_8. In Figure 4.6 (a), there are three bus components which form the U-shaped pattern — one vertical bus component connects modules m_1 and m_3, another vertical bus component connects modules m_2 and m_8, and the horizontal bus component connects modules m_1 and

m_2. If the overlap in the U-shaped pattern can be minimized, then total bus wirelength can be further optimized. Thus, the horizontal bus component is shifted down to minimize the overlap between the buses. Figure 4.6 (b) demonstrates the result of wirelength reduction. All the above steps are repeated until all horizontal and vertical bus components are searched.

Lemma 4.1 *Minimize the overlap in the obtained MST will further decrease the total bus wirelength of the MST.*

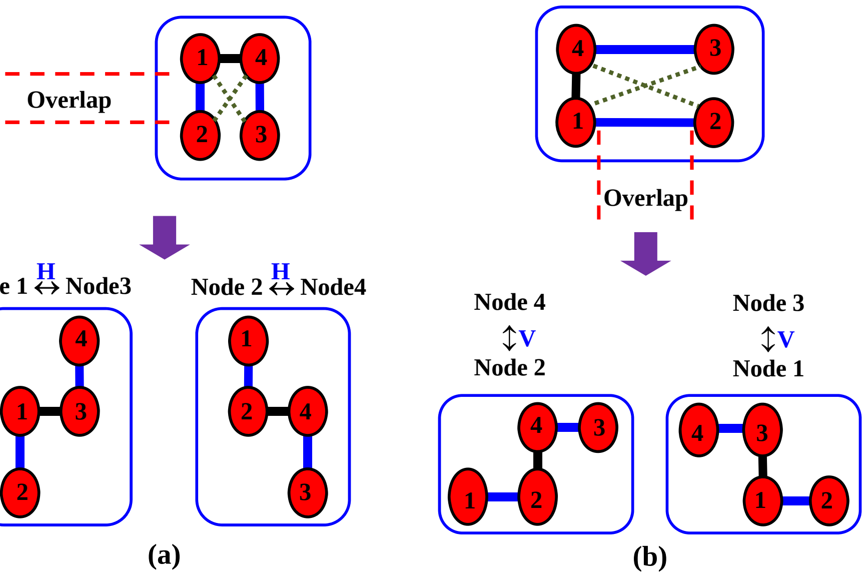

Figure 4.7: (a) Reduction rule for horizontal bus components. (b) Reduction rule for vertical bus components.

During constructing the MST for each bus, it ignores the actual information on the width and height of the module, as a result, the wirelength of the obtained MST may be further optimized. For each bus, it will construct a MST. In each MST, the bus modules are represented as the nodes, and the bus components are represented as the edges. If one MST contains the U-shaped bus pattern, it means that there is a overlap in the MST. Therefore, the bus wirelength can be minimized by minimizing the overlap in the MST.

Figure 4.7 shows the reduction rule. In Figure 4.7 (a), it shows the reduction rule for the horizontal bus components. Each time a matched U-shaped patten is found, it will check either the node 1 is horizontal with the node 3 or the node 2 is horizontal with the node 4. If the node 1 is horizontal with node 3, then the horizontal bus component connecting the node 1 and 4 is changed to connect the node 1 and node 3. If the node 2 is horizontal with node 4, then the horizontal bus component connecting the node 1 and 4 is changed to connect the node 2 and node 4. Then, the overlap is minimized to improve the bus wirelength. Figure 4.7 (b) shows the reduction rule for the vertical bus components. The rule can be derived similarly.

4.4.2 Coordinate Adjustment

When determining the coordinate of each bus, it only focuses on making each horizontal (vertical) buses feasible but not on minimizing the bus wirelength. However, different coordinates of each bus result in different interconnect wirelength between the horizontal and vertical bus components. Therefore, the bus wirelength can be further optimized by properly adjusting the coordinate of each bus component.

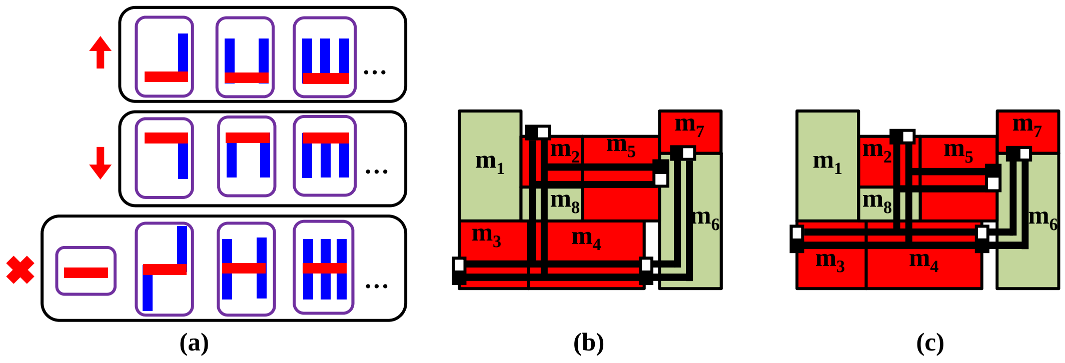

Figure 4.8: (a) Wirelength reduction rule. (b) Before wirelength reduction. (c) After wirelength reduction.

Figure 4.8 (a) illustrates how it works, the horizontal bus component is the target to be reduced the interconnect wirelength. In the first row, the number of the vertical bus components intersecting with the horizontal bus component in upper side are more than in lower side. Thus, it moves the horizontal bus component toward the upper side to improve its wirelength. In the second row, the number of the vertical bus components intersecting with the horizontal bus component in lower side are more than in upper side. Thus, it moves the horizontal bus component toward the lower side to improve its wirelength. In the third row, the number of the vertical bus components intersecting with the horizontal bus component are equal on both side, thus, the coordinate of the horizontal bus component is not changed.

In Figure 4.8 (b), the bus = $\{m_2, m_3, m_4, m_5, m_7\}$ is placed on the floorplan. Assume we first checks the horizontal bus $\{m_3, m_4\}$. According to the rule as shown in Figure 4.8 (a), it reduces the wirelength by moving the bus toward the upper side of the floorplan. The above steps are repeated until all horizontal and vertical buses are applied, the optimized result is demonstrated in Figure 4.8 (c).

Lemma 4.2 *Under some specific bus topologies, different positions of each horizontal (vertical) bus would result in different bus wirelength.*

As mentioned earlier, the coordinate of each horizontal bus component is assigned to the highest y-coordinate of all connected bus modules, and the coordinate of each vertical bus component is assigned to the highest x-coordinate of all connected bus modules. However, different coordinate of the bus components result in different interconnect wirelength between the horizontal and vertical bus components. If the coordinate of each bus component is not carefully assigned, then the total bus wirelength will be increased dramatically. Nevertheless, the task is more complicated if it determines the best position of all horizontal (vertical) bus components simultaneously. Therefore, a better approach is to decide an initial position for each horizontal (vertical) bus component, then it adjusts the coordinate of each bus component such that the better bus wirelength can be achieved.

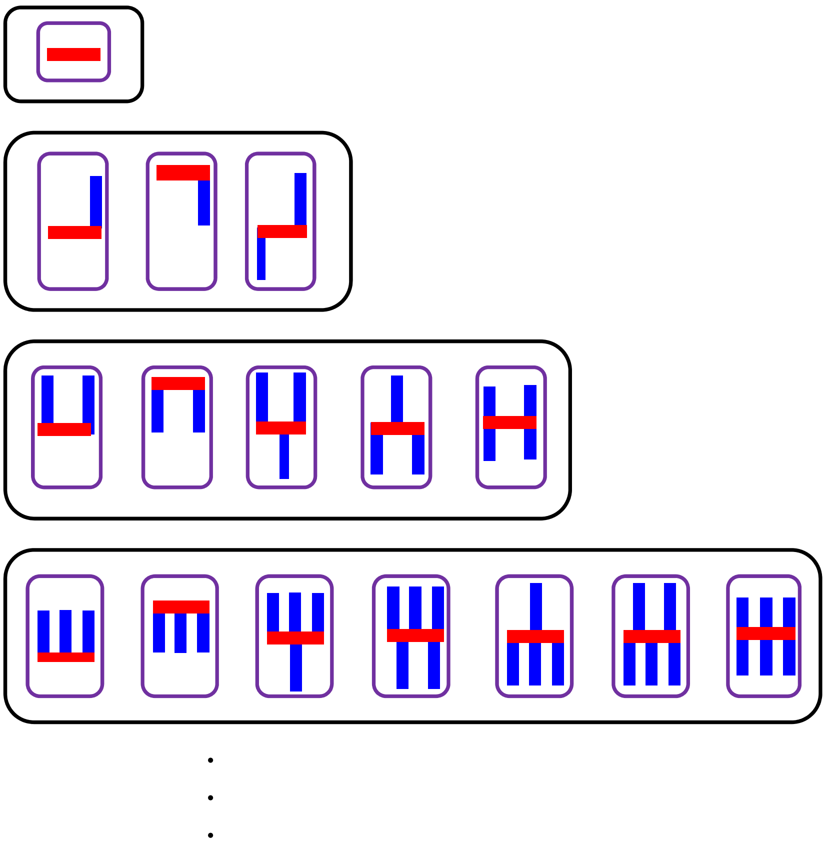

Figure 4.9: The original bus topology.

Figure 4.9 and Figure 4.10 illustrate how to modify the coordinate of each bus component. In this paper, we explore all possible connective situations between different horizontal and vertical bus components, parts of the possible connective situations are shown in Figure 4.9. In the first row, there is no any

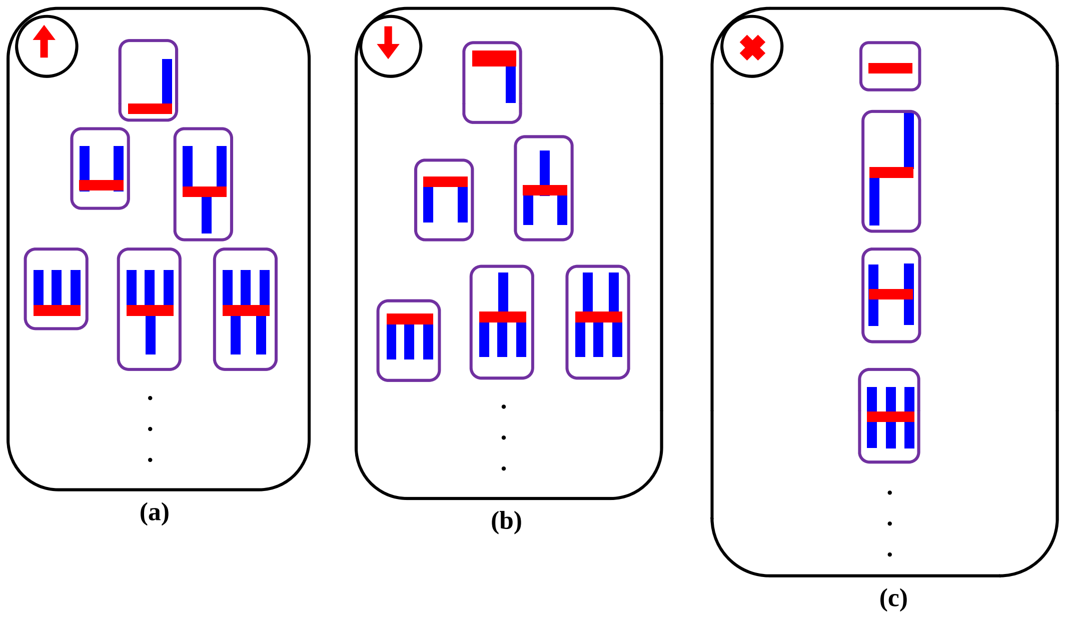

Figure 4.10: The rules for wirelength reduction.

vertical bus components intersecting with the horizontal bus component on both sides. In the second row, one vertical bus component intersects with the horizontal bus component on one side and one vertical bus component at most intersects with the horizontal bus component on the other side. In the third row, two vertical bus components intersect with the horizontal bus component on one side and two vertical bus components at most intersect with the horizontal bus component on the other side, the possible situations in other rows can be derived similarly. All the above situations in Figure 4.9 can be categorized into three cases as follows:

$Case1$: the number of intersecting vertical bus components on upper side of the horizontal bus component are more than on lower side as shown in Figure 4.10 (a), thus, the total bus wirelength will be improved by moving the horizontal bus component toward the upper side.

$Case2$: the number of intersecting vertical bus components on lower side of the horizontal bus component are more than on upper side as shown in Figure 4.10 (b), thus, the total bus wirelength will be optimized by moving the horizontal bus component toward the lower side.

*Case*3: the number of intersecting vertical bus components are equal on both sides of the horizontal bus component as shown in Figure 4.10 (c). No matter how the coordinate of the horizontal bus component is adjusted, the total bus wirelength remains unchanged.

Therefore, each time one horizontal bus component is given, the best bus position can be determined by the intersecting condition between the horizontal and vertical bus components. For any vertical bus components, the rule can be derived similarly.

4.5 Layer Assignment

In this paper, we explore the diagonal connection that makes the bus shape more flexible to increase the success rate. Since extra vias are ***forbidden*** at the bend of the diagonal buses, we develop an algorithm based on the graph coloring algorithm to assign the components of the diagonal buses to the same layer. However, graph coloring is computationally hard. It is an NP-complete problem if a given graph admits a k-coloring for k >= 3. In our formulation, it only reserves two layers for bus routing, then the layer assignment becomes 2-coloring problem and it can be solved in polynomial time. To obtain the overlapped information between the horizontal and vertical bus components, it first constructs the conflict graph by comparing the boundary and coordinate of each bus component. Each bus component is represented as a node in the graph, and the edges indicate two bus components are overlapped with each other. For minimizing the conflict between different bus components, it chooses the node that has the maximum degree to assign it to layer one, and all its neighbors in the graph are assigned to layer two. Then it continues another iteration by selecting one of the neighbors as the starting point. All the neighbors of the new starting point are assigned to the opposite layer. The above steps are repeated until all bus components are assigned to one of the two layers. If odd cycle occurs in the conflict graph, it means that some neighbors can not be assigned to the opposite layer of the starting point, then one of the buses is regarded as infeasible.

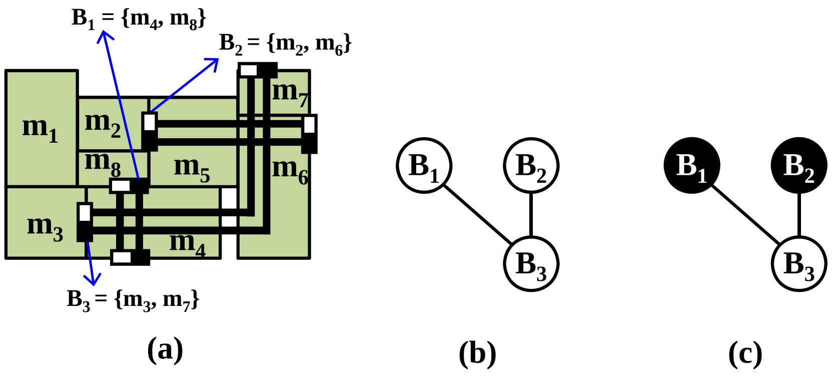

Figure 4.11: Layer assignment.

Figure 4.11 illustrates how the algorithm works. In Figure 4.11 (a), there are three buses and the modules passed by those buses are shown in the brace. The conflict graph is given in Figure 4.11 (b). Due to the bend of diagonal bus B_3 occurs at the module which is not a bus module, it must assign the bus components of the bus B_3 to the same layer. Thus, the diagonal bus B_3 is represented as the node B_3 in the graph. Initially, the node B_3 has the maximum degree in the conflict graph. We assign it to layer one, and all its neighbors B_1 and B_2 are assigned to layer two. Then it terminates because all buses are assigned to one of the two layers. Figure 4.11(c) is the result of layer assignment, the nodes in layer one are marked with white color, and the nodes in layer two are marked with black color.

4.6 Orientation Determination and Deviation Minimization

4.6.1 Deviation Minimization with Lookup Table Method

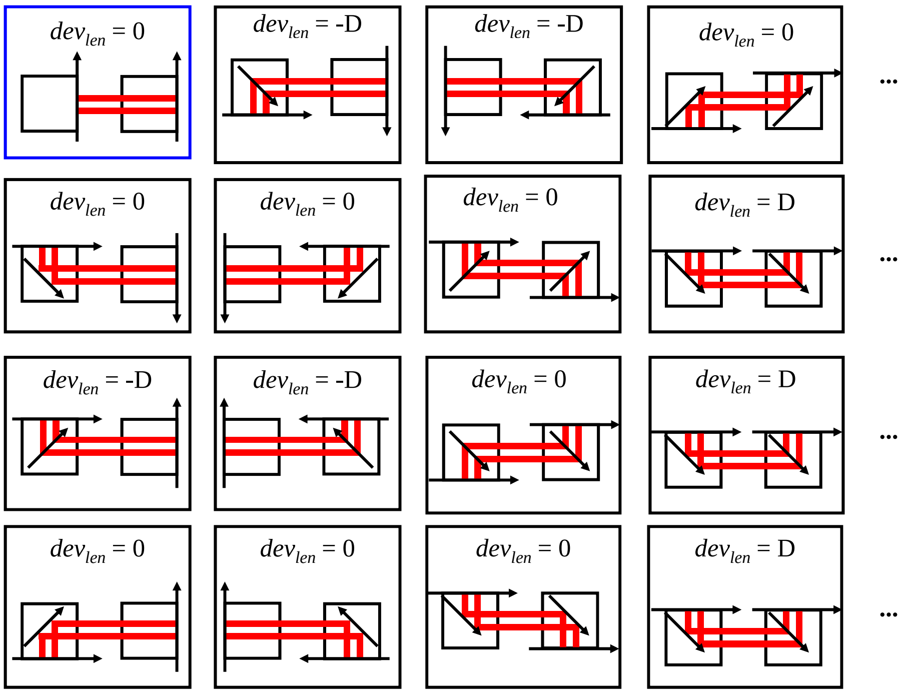

Figure 4.12: Total possible bus shapes between any two modules.

The bus routing problem has attracted much attention recently, and one of the popular problems is to determine the bus orientation and minimize the deviation at each load [12, 13]. However, these works mainly consider the impact of the blockages during bus routing that have different objective with ours. Besides, extra vias at the bend of each diagonal bus component is allowed in the above works, however, the vias have adverse effects on the bus delay, it is ***forbidden*** under our problem formulation. Therefore, we develop a new algorithm that can be integrated into our bus-driven floorplanner.

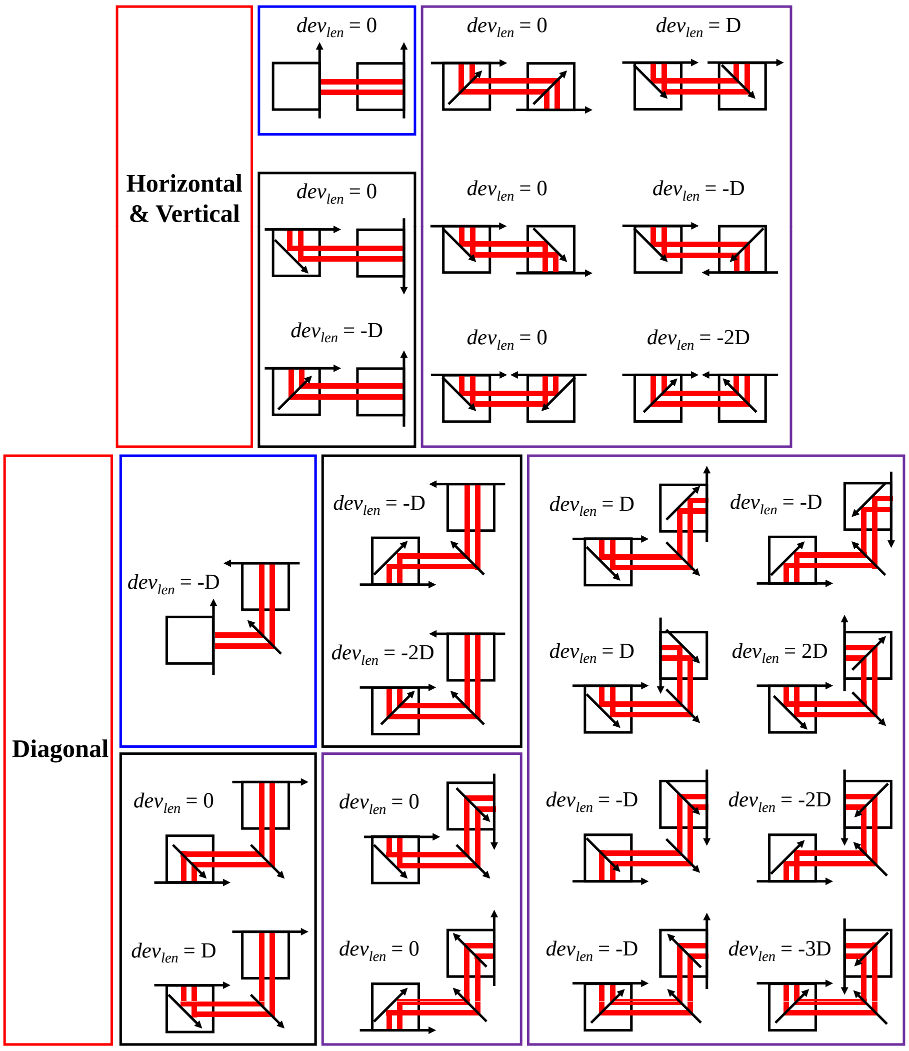

Figure 4.13: The bus patterns derived from the horizontal, vertical, and diagonal connections.

In this paper, we explore all possible bus shapes including the position of the bus pin between any two modules, and obtain total 150 possible bus shapes, part of the total possible bus shapes are illustrated in Figure 4.12. Different bus shapes can be mapped to some general patterns by mirroring or rotating, finally, total 24 bus patterns are concluded. Given the initial position of the bus pins on the modules, it can obtain the possible patterns.

Next, according to the accumulated deviation at previous module, it chooses the pattern holding the best accumulated deviation at the module. Finally, the accumulated deviation at the modules, the orientation and position of the bus pins, and the position of the bus pins can be determined. Figure 4.13 shows the 24 concluded patterns. The arrows represent the position and orientation of the bus pins at the turning nodes and the modules. Different cases can contribute different deviations.

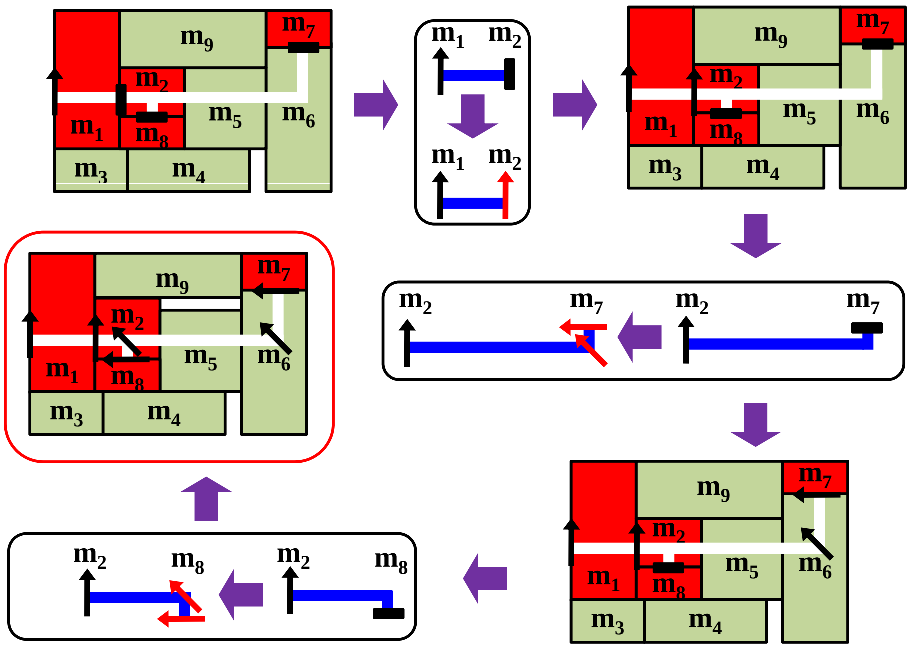

Figure 4.14: Determine the orientation and deviation for all bus pins and turning nodes.

Figure 4.14 illustrates how the algorithm works. There is a bus which passes through the modules in $\{m_1, m_2, m_7, m_8\}$, each module holds the initial position of the bus pins, and their orientation is not determined yet. We assume that m_1 is the driver, the searching order obtained from performing depth first search (DFS) on the MST is $m_1 \rightarrow m_2 \rightarrow m_7 \rightarrow m_8$. The deviation of the driver m_1 is 0. First, the module m_2 is picked as the candidate. The modules

m_1 and m_2 are connected with a horizontal bus component passing through the bus pins on the modules, so it can obtain one possible pattern from the 24 concluded patterns. The obtained pattern contributes 0 at the accumulated deviation, so the accumulated deviation from the driver to module m_2 is still 0, then the orientation of the bus pin on module m_2 is obtained.

Next, the module m_7 is chosen as the candidate. The modules m_2 and m_7 are connected with the diagonal bus component passing through the bus pins on the modules, so it can obtain one possible pattern from the 24 concluded patterns. The obtained pattern contributes $-$D at the accumulated deviation, so the accumulated deviation from the driver to the module m_7 is $-$D, then the orientations of the bus pin on the module m_7 and the turning node are obtained.

Finally, the module m_8 is selected as the candidate. The modules m_2 and m_8 are connected with only one vertical bus component. However, the bus pin on the module m_2 is not passed by the vertical bus component, so it can obtain two possible patterns from the 24 concluded patterns. One pattern contributes 0 at the accumulated deviation of the module, and another pattern contributed $-$D at the accumulated deviation of the module. Since the accumulated deviation from the driver to the previous module m_2 is 0, the pattern contributing 0 at the accumulated deviation of the module is chosen. Then the accumulated deviation from the driver to the module m_8 is 0, the orientations of the bus pin on the module m_8 and the turning node are obtained. Now the algorithm terminates because the position and orientation of all bus pins are determined.

4.6.2 Fast Deviation Update Based on Topology Comparison

To decrease the time for searching possible patterns, we propose another approach which divides the procedure of determining the deviation of each module into two steps — deviation determination and accumulated deviation (AD) update. In each iteration, the AD can be quickly updated after the deviation of the candidate is determined. Since the process of determining the deviation of each module is split, the shorter segment is needed for each comparison during searching the best pattern. If the bus pin on different modules can be connected by one straight line, then its deviation is equal to its AD, therefore, and those bus patterns are unnecessary kept in the concluded bus patterns. Finally, total 24 bus patterns can be further reduced to 10 types.

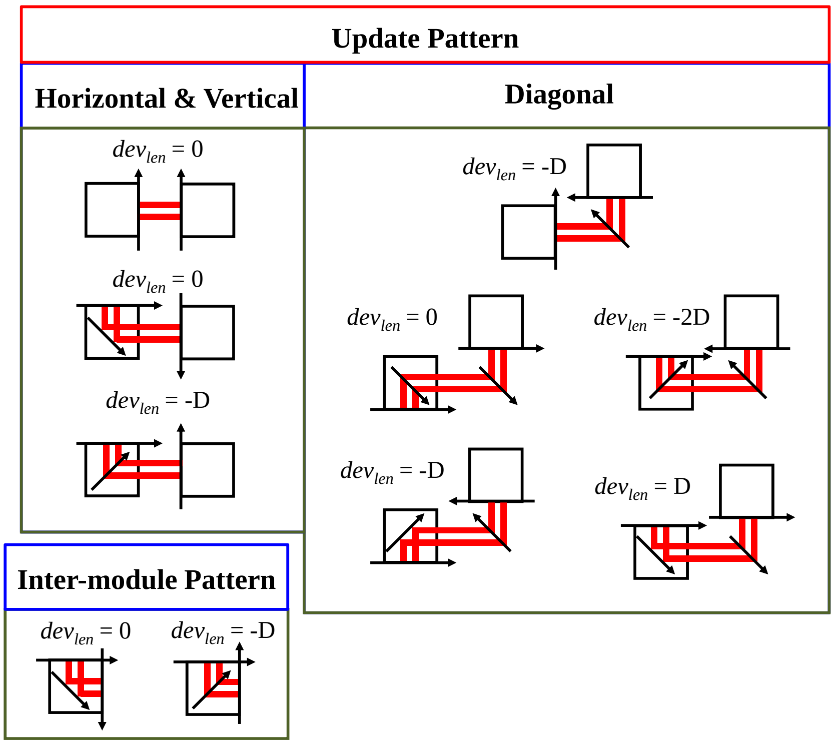

Figure 4.15: The reduced bus patterns.

Figure 4.15 gives total 10 concluded bus patterns. The deviation of each candidate is determined from the inter-module patterns, and the update patterns are used to update the AD of its neighbors. In this algorithm, the candidate is chosen based on the searching order which is obtained by performing breadth first search (BDF) on the MST, if a module has at least two neighbors, the neighbors are selected clockwise from its upper side.

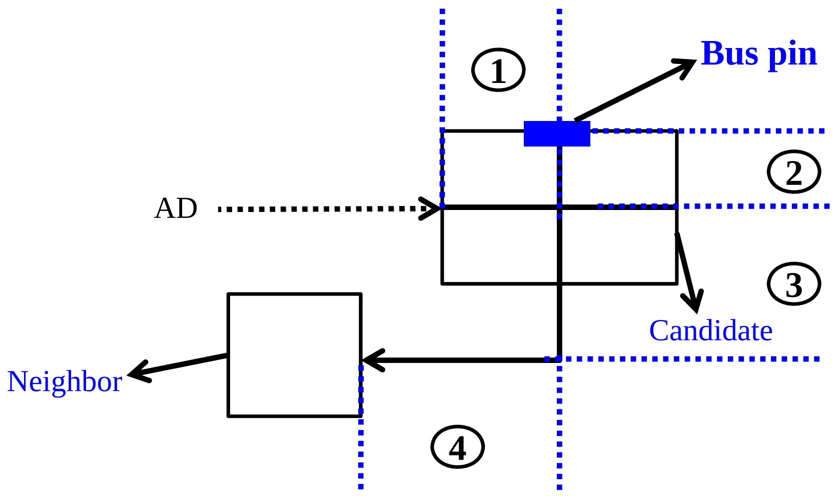

Figure 4.16: The procedure of deviation determination and AD update.

Figure 4.16 shows how the algorithm works. Initially, the bus pin of the candidate is placed on its upper boundary. Based on different AD on the module, the best deviation and orientation can be obtained from the inter-module patterns. We assume that the best deviation can be obtained at the candidate if the bus pin is placed on its upper boundary, then segment 1 and segment 2 are chosen to determine the deviation at the candidate. According to different relative positions between the candidate and its neighbors, it finds the matched pattern with segment 1, segment 3, and segment4 from the update patterns, then the AD of its neighbors can be obtained.

Figure 4.17 gives an example to illustrate how the algorithm works. There

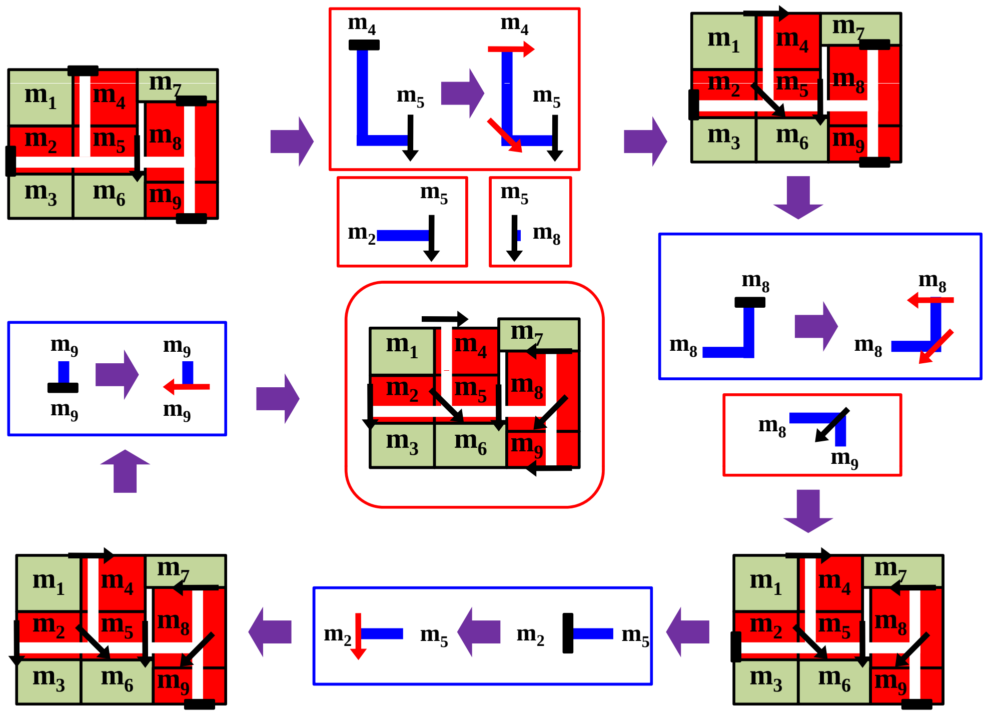

Figure 4.17: Determine the orientation and deviation for all bus pins and turning nodes.

is a bus which passes through the modules in $\{m_2, m_4, m_5, m_8, m_9\}$, each module holds the initial position of the bus pins, and their orientation is not determined yet. We assume that m_5 is the driver, the searching order obtained from performing BFS on the MST is $m_5 \to m_4 \to m_8 \to m_2 \to m_9$. First, the module m_5 is picked as the candidate. Since m_5 is the driver, its deviation is 0, then the AD of its neighbors are updated based on their relative positions. First, the best pattern from the update patterns is chosen to determine the orientation at the turning nodes in the driver, then the orientation and position of the bus pin on module m_4, the deviation at the module m_4 could be obtained. Next, it updates the AD for the modules m_8 and m_2. In next iteration, the module m_4 is selected as the candidate. However, the deviation at the module m_4 is obtained and it has no neighbor, the next module in the searching order is chosen to continue the procedure. In next iteration, the module m_8 is selected as the candidate. It first finds the best pattern from the inter-module patterns to determine its deviation. Then it updates AD for its neighbors m_9. In next iteration, the module m_2 is chosen as the candidate. Since the bus pin on module m_2 is connected by only one straight bus line, its deviation is equal to its AD. Finally, it selects the module m_9 as the candidate, the deviation of the module m_9 is equal to its AD because the bus pin is connected by only one straight bus line. Now the algorithm terminates because the position and orientation of all bus pins are determined.

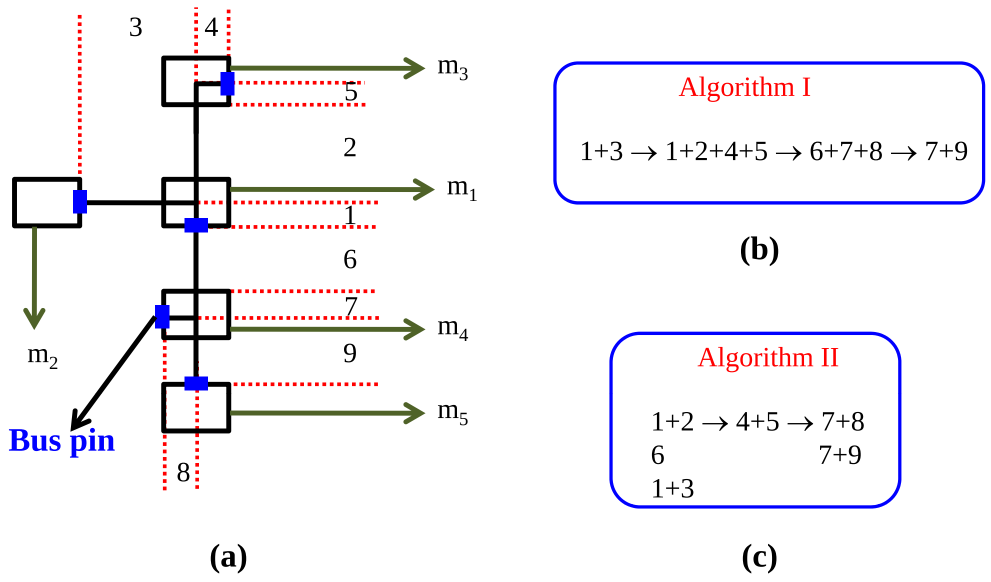

Figure 4.18: The difference between two algorithms.

We use Figure 4.18 to demonstrate the difference between two algorithms. In Figure 4.18 (a), there are five modules connected with one bus and each bus segment is marked as a number. The module m_1 is the driver. For the first algorithm, we assume the searching order is $m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow m_4 \rightarrow m_5$, the sequence of the bus segments which is processed during the algorithm is shown in Figure 4.18 (b). As for the second algorithm, the searching order is $m_1 \rightarrow m_3 \rightarrow m_4 \rightarrow m_2 \rightarrow m_5$, the sequence of the bus segments which is processed during the algorithm is shown in Figure 4.18 (c).

As given in Figure 4.18 (b) and Figure 4.18 (c), it shows that the bus patterns are more complicated in first algorithm than in second algorithm, thus, the number of the possible bus patterns in first algorithm are more than the number of the possible bus patterns in second algorithm, and it will spend more time to search for a best pattern.

Lemma 4.3 *For some specific bus topologies in the candidate, the AD of its neighbors can be directly calculated without searching for the best bus pattern.*

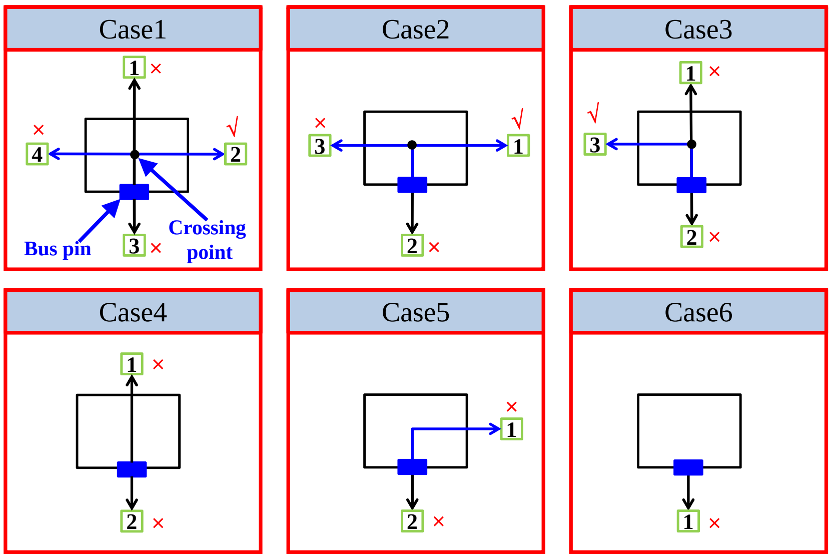

Figure 4.19: All possible topologies in driver.

In the following, several cases will be analyzed under different conditions. As shown in Figure 4.19, there are total six possible topologies in the driver, all other topologies can be mapped to these six cases by rotating or mirroring. In case 1, the driver has four neighbors. Since the modules at the direction 1 and direction 3 are connected by one straight line to the bus pin of the driver.

There is no deviation contributed by the driver. The AD of the module at the direction 1 and direction 3 are determined by the relation position between the driver and it. If the modules at the direction 1 and direction 3 are connected by one diagonal bus with the driver, then it will contribute D or -D to the AD of the module. The deviation of the modules at the direction 2 will be determined by finding the best pattern from the update patterns as shown in Figure 4.15, then the best orientation at the crossing point can be obtained. Since the orientation at the crossing point have been obtained, the AD of the module at the direction 4 can be calculated based on the orientation at the crossing point and the relative position between the driver and it. The directions are attached a tick mark if it needs to search the best bus pattern, and the directions are attached a cross mark if the AD of the module can be directly calculated. All other cases can be derived similarly.

Through the above cases, it can get the following conclusion: since the neighbors of the candidate are processed in the clockwise from the north side, if the bus pin on the driver is oriented horizontally and the driver has a right neighbor, then the deviation at the right neighbor is obtained by searching the best pattern from the update patterns, or the bus pin on the driver is oriented horizontally and the driver has only a left neighbor, then the deviation at the left neighbor is obtained by searching the best pattern from the update patterns. However, if the bus pin on the driver is oriented vertically and the driver has a upper neighbor, then the deviation at the upper neighbor is obtained by searching the best pattern from the update patterns, or the bus pin on the driver is oriented vertically and the driver has only a lower neighbor, then the deviation at the lower neighbor is obtained by searching the best pattern from the update patterns. After obtaining the orientation of the crossing point, the modules at other directions can be directly calculated based on the obtained orientation and the relative position between the driver and it.

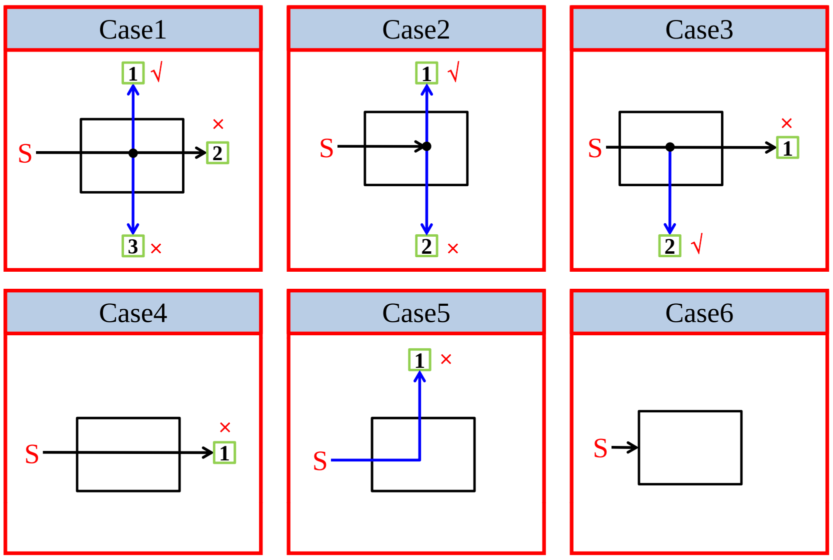

Figure 4.20: All possible topologies in load.

In Figure 4.20, there are total six possible topologies in the load. The "***S***" stands for the accumulated direction of the deviation from the driver. The rule can be derived in the same way when the candidate is the driver. If one neighbor of the candidate is at the first orthogonal direction compared with the accumulated direction and the orientation of the crossing point is still not determined, then the deviation of the neighbor is obtained by searching the best pattern from the update patterns, after that, the modules at other directions will be directly calculated based on the obtained orientation and the relative position between the driver and it. According to the above rules, it will avoid unnecessary comparison, then the time for searching the best deviation will be minimized.

4.7 Soft Module Adjustment

In order to minimize the chip area, it adjusts the dimension of some modules such that a better chip area can be obtained. The adjustment is the same as that in [2]. The step is processed with another simulated annealing process with the same cost function. In each iteration of the annealing process, it chooses the module lying on the critical path to change either its width or height a little bit. However, if any feasible bus become invalid after the adjustment, the candidate solution will be discarded.

Chapter 5

Experimental results

5.1 Experiments on Regular Testcases for [6] and [11]

TABLE I: TESTCASE STATISTICS OF TWO DATA SETS

	Data	No. of Modules	No. of Buses	Avg./Max. No. of Module on a Bus Net	The Bus Width of all Bus
Data set 1	ami33-a	33	5	3/4	10
	ami33-b	33	5	4/5	10
	ami33-c	33	5	5/6	10
	ami33-d	33	5	6/7	10
	ami33-e	33	5	7/8	10
	ami33-f	33	5	8/9	10
	ami49-a	49	1	10/10	150
	ami49-b	49	1	20/20	150
	ami49-c	49	1	30/30	150
	ami49-d	49	1	40/40	150
	ami49-e	49	1	49/49	150
Data set 2	ami33-g	33	3	2/2	55
	ami33-h	33	3	5/5	55
	ami33-i	33	3	7/7	55
	ami49-f	49	3	5/5	150, 300, 450
	ami49-g	49	3	6/6	150, 300, 450
	ami49-h	49	3	7/7	150, 300, 450

In this section, we perform the experiments on the algorithm which is published on GLSVLSI 2010 with the state-of-the-art bus-driven floorplanner [6] which is published on ASPDAC 2008. We implemented our bus-pin-aware bus-driven floorplanner in C language on a 1.86-GHz Linux machine with 2GB memory. All testcases are run by both our floorplanner and the bus-driven floorplanner [6] on the same machine. For fair comparison, each floorplanner runs ten times for each testcase and the average is reported. Since the impact of the bus pin is not considered in [6], the bus pin of each module is not defined

TABLE II: COMPARISON BETWEEN [6] AND [11] ON TWO DATA SETS

Data	[6]					[11]						
	Time* (s)	No Soft Adjust (%)	With Soft Adjust (%)	Wirelength	Success Rate** (%)	Time (s)	No Soft Adjust (%)	With Soft Adjust (%)	Wirelength	Success Rate (%)	Deviation (D)***	
											General****	Optimization
Data set 1												
ami33-a	13.60 (+1)	6.30%	1.42	3157.30	92.10	4.11 (+0)	3.64	1.56	710.50	99.80	0.29	0.00
ami33-b	15.80 (+0)	6.59%	1.84	4282.10	84.60	6.13 (+0)	3.97	1.67	1366.70	99.28	0.59	0.02
ami33-c	18.10 (+0)	7.82%	2.34	5110.30	73.70	7.35 (+0)	4.58	1.72	2433.20	98.18	0.69	0.04
ami33-d	22.10 (+1)	8.36%	2.35	5633.20	71.70	8.05 (+0)	4.76	1.87	3318.40	97.81	0.82	0.04
ami33-e	60.40 (+2)	8.99%	1.97	6225.40	75.60	8.91 (+0)	4.59	1.88	4578.60	95.46	0.83	0.05
ami33-f	66.00 (+1)	9.46%	1.98	6861.80	69.60	10.01 (+0)	4.68	1.98	5087.20	95.12	1.32	0.06
ami49-a	24.60 (+2)	5.32%	1.21	11568.70	91.78	8.96 (+0)	2.80	0.86	5304.40	100.00	1.27	0.01
ami49-b	38.60 (+2)	5.97%	1.55	16413.90	89.36	10.14 (+0)	3.35	0.95	8189.40	100.00	1.59	0.09
ami49-c	38.10 (+2)	6.47%	1.50	19841.00	84.86	11.84 (+1)	3.96	1.30	12761.60	100.00	2.29	0.15
ami49-d	38.70 (+2)	7.86%	1.74	NA	81.02	15.96 (+1)	4.71	1.56	16955.80	100.00	3.32	0.17
ami49-e	52.00 (+2)	9.35%	2.16	NA	77.82	20.86 (+1)	5.23	1.55	22452.60	100.00	5.24	0.23
Average	347%	178%	119%	181%	82%	100%	100%	100%	100%	100%	2135%	100%
Data set 2												
ami33-g	6.80 (+1)	5.36	1.47	433.60	80.12	5.65 (+0)	2.36	1.28	118.30	95.58	0.33	0.20
ami33-h	10.70 (+1)	7.16	1.87	2654.70	72.90	6.51 (+0)	3.76	1.51	1999.30	91.30	0.78	0.08
ami33-i	24.10 (+1)	6.68	2.13	3778.80	52.24	7.16 (+0)	3.84	1.89	3146.80	90.14	0.94	0.06
ami49-f	40.90 (+2)	6.11	1.60	17644.00	1.21	9.75 (+1)	3.81	1.16	11716.40	89.05	0.89	0.09
ami49-g	42.50 (+2)	6.69	3.58	30333.00	2.69	9.80 (+1)	3.86	1.10	12289.70	88.88	0.85	0.09
ami49-h	52.20 (+2)	7.01	1.15	22318.00	3.21	10.09 (+1)	4.20	1.23	15646.80	88.71	0.84	0.07
Average	358%	179%	144%	172%	39%	100%	100%	100%	100%	100%	781%	100%

*Figures inside brackets represent the run time for soft module adjustment.
**The percentage of the floorplan generated by all iterations of the SA in which all the buses are feasible.
****The deviation (D) is obtained from total deviations at each load divided by total number of loads.
***The deviation at each load is estimated by total number of turning nodes from the driver to the load divided by total number of loads.

in the data set 1. Therefore, we modify those testcases and assign the initial position of each bus pin randomly.

In addition to perform experiments on data set 1 used in [6], we also create data set 2 by modifying some testcases in data set 1 to demonstrate the excellent routability of our algorithm. In data set 2, the bus width of each bus is wider than data set 1, i.e., it is more difficult to obtain a feasible solution. In each testcase, the first module of each bus is treated as the driver. The details of all testcases are shown in Table I.

The first experiment is performed on data set 1, and the comparison results are shown in Table II. As for the experiments about the deviation of each module, for fair comparison, we only list the average deviation (D) in the rightmost two columns for estimated and optimized bus routing results from our floorplanner because [6] does not consider the impact of the bus pins. In this paper, we consider the diagonal connection between two modules, it makes the bus shape more flexible, and the size of the solution space is increased such that a better solution can be obtained. Compared with [6], the experimental results show that our algorithm performs better in runtime by 3.47×, success rate by 1.22×, and reduced the deadspace by 1.19×. We also develop an algorithm to reduce the wirelength, the experimental results show that our algorithm performs better in wirelength by 1.81×. By considering the accumulated deviation during determining the position and orientation of each bus pin, it can obtain better deviation at each load. Experimental results show that the optimized deviation achieves 21.35× better than the estimated bus routing solution on average. In our platform, [6] does not generate a feasible final solution in ami49-d and ami49-e, these testcases are ignored in wirelength comparison. In other testcases, [6] fails to generate a feasible final solution in some iterations of the ten annealing processes, these iterations are also ignored in wirelength comparison.

The experimental results of data set 2 are also shown in Table II. Compared with [6], the experimental results show that our algorithm performs better in runtime by 3.58×, success rate by 2.56×, and reduced the deadspace by 1.44×. In terms of wirelength, our algorithm performs better by 1.72×. With deviation optimization, the deviation achieves 7.81× better than the estimated ones on average. For ami49-f, ami49-g, and ami49-h, [6] could not generate the feasible solution effectively. However, our floorplanner still holds high solution quality in all aspects. Some packing results are shown in Figure 5.1.

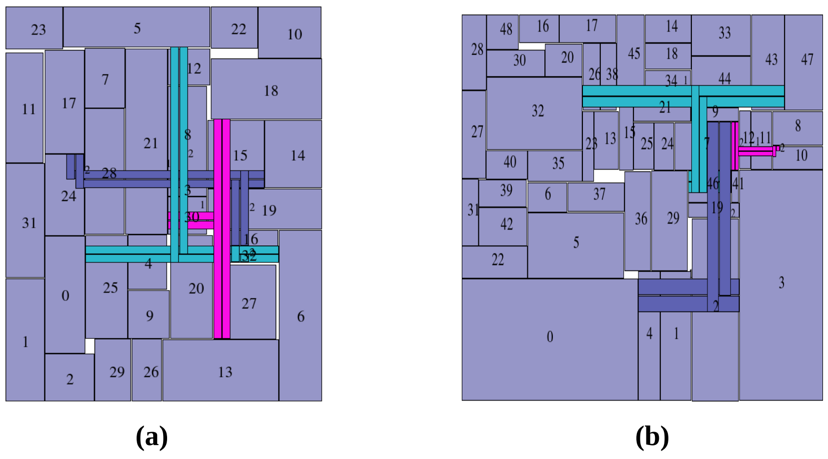

Figure 5.1: (a) Packing result of ami33-i. The buses are {13, 16, 18, 19, 21, 27, 30}, {0, 5, 6, 20, 25, 30, 32}, {14, 15, 16, 17, 24, 28, 32}. (b) Packing result of ami49-h. The buses are {7, 8, 9, 10, 11, 12, 41}, {29, 32, 41, 43, 44, 46, 47}, {0, 1, 2, 3, 4, 9, 46}.

5.2 The Comparison Between [11] and This Work

In the remaining experiments, we are implemented our bus-driven floorplanner in C language on a 2-GHz Linux machine with 16GB memory. Since the work which is published on GLSVLSI 2010 only performs the algorithm described in Section 4.6.1 after the SA, the deviation of each testcase could not be minimized effectively. In this paper, we develop another efficient approach to determine the deviation of each module such that it can be integrated into the simulated annealing process. The comparison results are shown in Table III. Since the effect of the deviation is considered in this work during SA, i.e., the solution space is much more constrained. The run time is longer than GLSVLSI 2010 and the deadspace and wirelength are increased, however, the deviation for each testcase can be further optimized to 0 as shown in the experimental results.

TABLE III: COMPARISON BETWEEN [11] AND THIS WORK

Data	[11]					This work					Deviation (D)		
	Time (s)	No Soft Adjust (%)	With Soft Adjust (%)	Wirelength	Success Rate (%)	Time (s)	No Soft Adjust (%)	With Soft Adjust (%)	Wirelength	Success Rate (%)	1^{*}	2^{**}	3^{***}
ami33-a	4.11 (+0)	3.64	1.56	710.50	99.80	5.18 (+0)	4.16	1.67	703.15	99.64	0.29	0.00	0.00
ami33-b	6.13 (+0)	3.97	1.67	1366.70	99.28	7.26 (+0)	4.38	1.86	1669.13	98.68	0.59	0.02	0.00
ami33-c	7.35 (+0)	4.58	1.72	2433.20	98.18	8.61 (+0)	5.36	1.86	2648.46	97.17	0.69	0.04	0.00
ami33-d	8.05 (+0)	4.76	1.87	3318.40	97.81	9.94 (+0)	5.89	1.88	3509.56	96.68	0.82	0.04	0.00
ami33-e	8.91 (+0)	4.59	1.88	4578.60	95.46	12.11 (+0)	6.77	1.94	4947.38	94.47	0.83	0.05	0.00
ami33-f	10.01(+1)	4.68	1.98	5087.20	95.12	16.03 (+1)	7.70	2.01	5768.17	93.68	1.32	0.06	0.00
ami33-g	5.65 (+0)	2.36	1.28	118.30	95.58	5.80 (+0)	2.64	1.46	173.82	94.58	0.33	0.20	0.00
ami33-h	6.51 (+0)	3.76	1.51	1999.30	91.30	6.74 (+0)	4.12	1.80	2160.46	92.65	0.78	0.08	0.00
ami33-i	7.16 (+0)	3.84	1.89	3146.80	90.14	7.84 (+0)	4.71	2.26	3483.73	89.72	0.94	0.06	0.00
ami49-a	8.96 (+0)	2.80	0.86	5304.40	100.00	9.68 (+0)	2.93	1.06	6296.91	100.00	1.27	0.01	0.00
ami49-b	10.14 (+1)	3.35	0.95	8189.40	100.00	10.50 (+1)	4.24	1.10	9437.97	100.00	1.59	0.09	0.00
ami49-c	11.84 (+1)	3.96	1.30	12761.60	100.00	14.12 (+1)	5.06	1.40	13705.59	100.00	2.29	0.15	0.00
ami49-d	15.96 (+1)	4.71	1.56	16955.80	100.00	21.02 (+1)	6.72	1.60	18017.65	100.00	3.32	0.17	0.00
ami49-e	20.86 (+1)	5.23	1.55	22452.60	100.00	33.85 (+1)	7.89	1.88	23791.37	100.00	5.24	0.23	0.00
ami49-f	9.75 (+1)	3.81	1.16	11716.40	89.05	9.87 (+1)	4.04	1.52	12121.36	87.73	0.89	0.09	0.00
ami49-g	9.80 (+1)	3.86	1.10	12289.70	88.88	10.10 (+1)	4.08	1.72	13268.83	87.25	0.85	0.09	0.00
ami49-h	10.09 (+1)	4.20	1.23	15646.80	88.71	11.29 (+1)	4.46	1.79	16318.52	87.17	0.84	0.07	0.00
Average	81%	80%	87%	93%	101%	100%	100%	100%	100%	100%	-	93.67%	100%

*The deviation at each load is estimated by total number of turning nodes from the driver to the load divided by total number of loads.
**The deviation (D) is obtained by the algorithm which is published in GLSVLSI 2010.
***The deviation (D) is obtained by the algorithm which is developed in this work.

5.3 The Comparison Between Modified [11] and This Work

In this section, we modify the algorithm which is published in GLSVLSI 2010 such that it has the same structure with this work, i.e., the effect of deviation is considered during SA. The difference between the modified algorithm and this work is that the algorithm described in Section 4.6.1 is performed to determine the orientation of each bus pin in the modified algorithm, and the algorithm described in 4.6.2 is performed to determine the orientation of each bus pin in this work. The comparison results are shown in Table IV. The results show that deadspace, wirelength, success rate, and deviation is nearly the same, nevertheless, the algorithm proposed in this paper is run faster than the modified method. Therefore, the proposed approach is more suitable on optimizing the deviation during the SA.

TABLE IV: COMPARISON BETWEEN MODIFIED [11] AND THIS WORK

Data	Modified [11]						This work					
	Time (s)	No Soft Adjust (%)	With Soft Adjust (%)	Wirelength	Success Rate (%)	Deviation (D)	Time (s)	No Soft Adjust (%)	With Soft Adjust (%)	Wirelength	Success Rate (%)	Deviation (D)
ami33-a	7.42 (+0)	4.14	1.79	681.69	99.70	0.00	5.18 (+0)	4.16	1.67	703.15	99.64	0.00
ami33-b	8.87 (+0)	4.70	1.75	1658.24	98.91	0.00	7.26 (+0)	4.38	1.86	1669.13	98.68	0.00
ami33-c	9.91 (+0)	5.32	1.95	2604.68	97.54	0.00	8.61 (+0)	5.36	1.86	2648.46	97.17	0.00
ami33-d	11.41 (+0)	5.87	1.78	3495.11	97.23	0.00	9.94 (+0)	5.89	1.88	3509.56	96.68	0.00
ami33-e	14.37 (+1)	6.91	1.99	5063.59	94.23	0.00	12.11 (+0)	6.77	1.94	4947.38	94.47	0.00
ami33-f	17.39 (+2)	7.65	1.87	5777.51	93.65	0.00	16.03 (+1)	7.70	2.01	5768.17	93.68	0.00
ami33-g	6.64 (+0)	2.60	1.48	219.99	94.34	0.00	5.80 (+0)	2.64	1.46	173.82	94.58	0.00
ami33-h	8.14 (+0)	4.14	1.81	2111.41	92.76	0.00	6.74 (+0)	4.12	1.80	2160.46	92.65	0.00
ami33-i	9.88 (+0)	4.76	2.13	3508.04	90.48	0.00	7.84 (+0)	4.71	2.26	3483.73	89.72	0.00
ami49-a	10.69 (+1)	2.99	0.96	6142.12	100.00	0.00	9.68 (+0)	2.93	1.06	6296.91	100.00	0.00
ami49-b	11.71 (+1)	4.15	1.09	9508.81	100.00	0.00	10.50 (+1)	4.24	1.10	9437.97	100.00	0.00
ami49-c	15.97 (+1)	4.96	1.37	13771.02	100.00	0.00	14.12 (+1)	5.06	1.40	13705.59	100.00	0.00
ami49-d	23.40 (+1)	6.47	1.72	18458.61	100.00	0.00	21.02 (+1)	6.72	1.60	18017.65	100.00	0.00
ami49-e	35.60 (+1)	8.04	1.80	22824.60	100.00	0.00	33.85 (+1)	7.89	1.88	23791.37	100.00	0.00
ami49-f	11.14 (+1)	4.08	1.58	12365.64	87.78	0.00	9.87 (+1)	4.04	1.52	12121.36	87.73	0.00
ami49-g	11.22 (+1)	4.25	1.87	13086.74	87.55	0.00	10.10 (+1)	4.08	1.72	13268.83	87.25	0.00
ami49-h	12.45 (+1)	4.51	1.73	16058.68	86.76	0.00	11.29 (+1)	4.46	1.79	16318.52	87.17	0.00
Average	114%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

5.4 Experiments on Larger Testcases for [11] and This Work

In this section, we will perform the experiments on larger testcases for the algorithm which is published in GLSVLSI 2010 and this work. The set of testcases are derived from n100, n200, n300 with bus sizes ranging from 10 to 40. The details of all testcases are listed in Table V.

TABLE V: LARGER TESTCASE STATISTICS

	Data	No. of Modules	No. of Buses	Avg./Max. No. of Module on a Bus Net	The Bus Width of all Bus
Data set	n100-a	100	5	10/10	5
	n100-b	100	5	12/12	5
	n100-c	100	5	14/14	5
	n100-d	100	5	16/16	5
	n100-e	100	5	18/18	5
	n100-f	100	5	20/20	5
	n200-a	200	5	20/20	5
	n200-b	200	5	25/25	5
	n200-c	200	5	30/30	5
	n200-d	200	5	35/35	5
	n200-e	200	5	40/40	5
	n300-a	300	5	10/10	5
	n300-b	300	5	20/20	5
	n300-c	300	5	30/30	5

To the best of our knowledge, no previous works can handle the buses with

TABLE VI: COMPARISON BETWEEN [11] AND THIS WORK ON LARGER TESTCASES

Data	[11]						This work					
	Time (s)	No Soft Adjust (%)	With Soft Adjust (%)	Wirelength	Success Rate (%)	Deviation (D)	Time (s)	No Soft Adjust (%)	With Soft Adjust (%)	Wirelength	Success Rate (%)	Deviation (D)
n100-a	28.63 (+2)	5.44	4.15	2059.33	92.90	0.08	26.80 (+2)	4.82	3.75	1798.30	94.98	0.00
n100-b	30.41 (+1)	5.02	4.00	2302.40	92.76	0.12	30.20 (+2)	5.38	3.71	2530.96	93.73	0.00
n100-c	34.56 (+1)	6.04	4.36	2755.86	91.83	0.12	36.80 (+2)	6.09	4.05	2930.51	92.19	0.00
n100-d	39.31 (+2)	6.46	4.90	3245.32	92.05	0.15	45.40 (+3)	6.42	4.15	3453.50	88.74	0.00
n100-e	45.54 (+2)	5.90	4.90	3801.20	90.95	0.14	56.80 (+3)	6.87	4.42	3998.31	90.57	0.00
n100-f	51.05 (+2)	6.33	4.92	4379.96	87.27	0.16	69.40 (+3)	7.65	4.79	4537.09	86.68	0.00
n200-a	129.05 (+3)	7.45	6.61	3216.03	90.66	0.15	146.60 (+6)	7.64	6.08	3381.05	88.13	0.00
n200-b	196.60 (+6)	8.27	6.96	3873.05	81.32	0.15	256.40 (+8)	9.19	6.76	4220.13	82.07	0.00
n200-c	366.40 (+4)	9.24	8.45	4728.52	62.61	0.17	406.60 (+9)	9.87	7.04	5334.82	76.68	0.00
n200-d	511.80 (+12)	9.65	8.50	5547.33	69.07	0.18	672.80 (+13)	10.44	7.43	5848.39	76.75	0.01
n200-e	581.60 (+7)	9.69	8.90	6566.78	75.29	0.17	1051.60 (+23)	11.93	7.85	7200.74	69.72	0.02
n300-a	143.20 (+5)	7.60	6.91	1634.43	65.35	0.12	120.40 (+9)	7.34	6.53	1881.85	74.72	0.00
n300-b	251.80 (+7)	8.34	7.77	3892.61	73.26	0.15	270.20 (+12)	8.70	7.52	4329.88	80.46	0.04
n300-c	815.20 (+10)	10.14	9.45	5654.37	49.34	0.21	760.60 (+16)	9.53	7.71	6180.13	59.26	0.04
Average	82%	94%	111%	93%	97%	1664%	100%	100%	100%	100%	100%	100%

such a large net size. The comparison results are shown in Table VI. The experimental results show that our algorithm performs better in success rate by 1.03×, deviation by 16.64×, and reduces the deadspace 1.11×. Since it has to minimize the deviation during SA, it constrains the solution space such that the bus wirelength are increased and spends more time on adjusting the bus shape to obtain a better solution. However, the deviation can be further optimized to 0 in lots of the testcases, and the deviation is effectively minimized in the difficult testcases. Some packing results are shown in Figure 5.2.

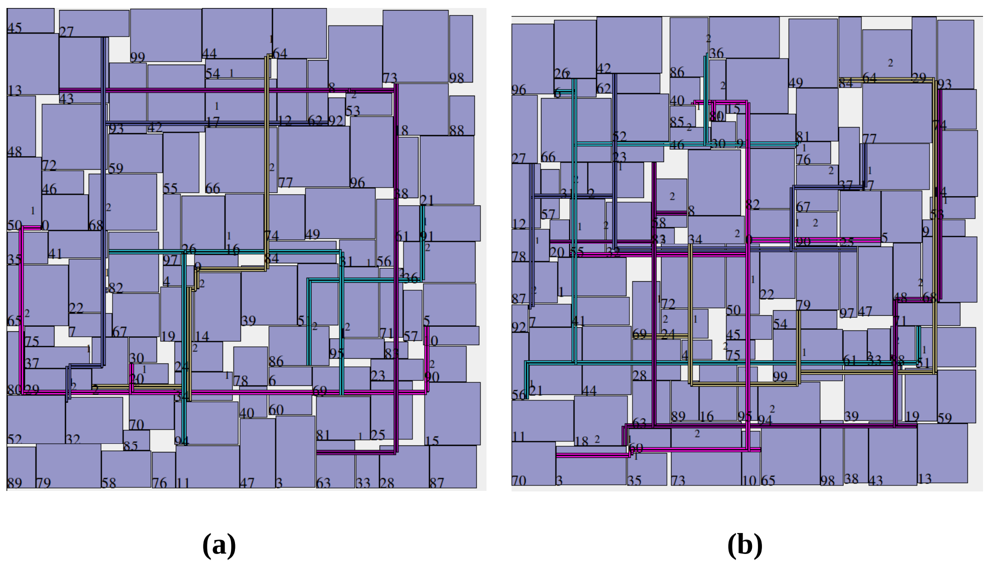

Figure 5.2: (a) Packing result of n100-c. The buses are {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 5, 15, 25, 35}, {1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 6, 16, 26, 36}, {2, 12, 22, 32, 42, 52, 62, 72, 82, 92, 7, 17, 27, 37}, {3, 13, 23, 33, 43, 53, 63, 73, 83, 93, 8, 18, 28, 38}, {4, 14, 24, 34, 44, 54, 64, 74, 84, 94, 9, 19, 29, 39}. (b) Packing result of n100-f. The buses are {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 5, 15, 25, 35, 45, 55, 65, 75, 85, 95}, {1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 6, 16, 26, 36, 46, 56, 66, 76, 86, 96}, {2, 12, 22, 32, 42, 52, 62, 72, 82, 92, 7, 17, 27, 37, 47, 57, 67, 77, 87, 97}, {3, 13, 23, 33, 43, 53, 63, 73, 83, 93, 8, 18, 28, 38, 48, 58, 68, 78, 88, 98}, {4, 14, 24, 34, 44, 54, 64, 74, 84, 94, 9, 19, 29, 39, 49, 59, 69, 79, 89, 99}.

Chapter 6

Conclusions

As the early stage of physical design, the floorplanning quality has dramatic impacts on the chip performance. In this work, we proposed a high-quality bus-driven floorplanning algorithm considering the practical impacts of the bus pins. The experimental results show that our floorplanner outperforms than the state-of-the-art floorplanner [6] in all aspects. Besides, our algorithm can effectively minimize the deviation in larger testcases. The future work lies in extending our algorithm to handle other practical constraints such as fixed-outline, thermal effect, and etc.

Chapter 7

Acknowledge

The authors would like to thank Tilen Ma and Prof. Evangeline F.Y. Young of the Chinese University of Hong Kong for providing the authors with benchmarks and program for the comparative studies.

Bibliography

[1] F. Rafiq, M. Chrzanowska-Jeske, H. H. Yang, and N. Sherwani, “Integrated floorplanning with buffer/channel insertion for bus-based microprocessor designs,” *Proc. of ISPD*, pp. 56–61, 2002.

[2] H. Xiang, X. Tang, and Martin D. F. Wong, “Bus-driven floorplanning,” *Proc. of ICCAD*, pp. 66–73, 2003.

[3] T. C. Chen and Y. W. Chang, “Modern floorplanning based on fast simulated annealing,” *Proc. of ISPD*, pp. 104–112, 2005.

[4] J. H. Y. Law and E. F. Y. Young, “Multi-bend bus-driven floorplanning,” *Proc. of ISPD*, pp.113–120, 2005.

[5] H. Xiang, L. Deng, L.D. Huang, Martin D. F. Wong, “OPC-friendly bus driven floorplanning,” *Proc. of ISQED*, pp. 847–852, 2007.

[6] T. Ma and E. F. Y. Young, “TCG-based multi-bend bus-driven floorplanning,” *Proc. of ASPDAC*, pp. 192–197, 2008.

[7] D. H. Kim and S. K. Lim, “Global bus route optimization with application to microarchitectural design exploration,” *Proc. of ICCD*, pp. 658–663, 2008.

[8] D. H. Kim and S. K. Lim, “Bus-aware microarchitectural floorplanning,” *Proc. of ASPDAC*, pp. 204–208, 2008.

[9] W. Sheng, S. Dong, Y. Wu, and S. Goto, “Fixed outline multi-bend bus driven floorplanning,” *Proc. of ISQED*, pp. 632–637, 2010.

[10] O. He, S. Dong, J. Bian, S. Goto, and C.K. Cheng, "Bus via reduction based on floorplan revising," *Proc. of GLSVLSI*, pp. 9–14, 2010.

[11] B. S. Wu and T. Y. Ho, "Bus-pin-aware bus-driven floorplanning," *Proc. of GLSVLSI*, pp. 27–32, 2010.

[12] F. Mo and R. K. Brayton, "Semi-detailed bus routing with variation reduction," *Proc. of ISPD*, pp. 143–150, 2007.

[13] F. Mo and R. K. Brayton, "A simultaneous bus orientation and bused pin flipping algorithm," *Proc. of ICCAD*, pp. 386–389, 2007.

[14] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing-based module placement," *Proc. of ICCAD*, pp. 472–479, 1995.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms," *2nd ed., MIT Press and McGraw-Hill Book Co.*, 2001.