An Effective Netlist Planning Approach for Double-sided Signal Routing*

Tzu-Chuan Lin, Fang-Yu Hsu, Wai-Kei Mak, and Ting-Chi Wang
Department of Computer Science, National Tsing Hua University, Taiwan

2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3503-9354-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASP-DAC58780.2024.10473897

***Abstract*—Separating the power delivery network (PDN) from the front-side metal stack and using the back-side metal stack primarily for the PDN has been proposed to improve the PDN performance. To well utilize the surplus routing resources left on the back side after the PDN is built, we study in this paper how to route signal nets on both front and back sides (i.e. double-sided signal routing). To this end, we present a netlist planning approach that is able to well distribute a set of signal nets to two sides and properly insert a set of bridging cells such that after performing placement legalization and signal routing on each side separately, a high-quality double-sided routing solution can be produced. Our netlist planning approach has been combined with a commercial place and route tool, and our experimental results show that compared to traditional single-sided routing without back-side PDN, double-sided routing achieves 9.1% reduction in wirelength and 1.8% decrease in via count. Additionally, for critical nets, the percentage of the total length of their wire segments routed on preferred metal layers were improved by 13.6%.**

***Index Terms*—back-side power delivery network, double-sided signal routing**

I. Introduction

The traditional approach of constructing the power delivery network (PDN) in the standard back-end-of-line (also called front-side metal stack) of a chip has limitations, especially in terms of IR drop, which becomes a bottleneck for high-performance and low-supply-voltage designs. It forces designers to reserve front-side routing resources to construct a larger and more robust PDN. To address this issue, the concept of back-side PDN has been proposed. Here are two instances of recent technologies for back-side PDN. Imec presented an approach that involves integrating power rails into the shallow trench isolation and silicon metallization scheme on the back side [1]. It allows for efficient power delivery while utilizing the back-side resources. Intel proposed PowerVia, which was implemented on the Intel 4 process [2]. PowerVia enables relaxed scaling of the front-side metal layers, providing benefits in terms of power delivery and performance.

Currently, the back-side resources of a chip are dedicated to the PDN, while the signal nets are routed separately using the front-side metal layers that are not affected by the PDN. However, these back-side routing resources have the potential to provide faster transmission speed with lower resistance. If the back-side metal resources that are not used by the PDN can be utilized for routing signal nets, there is an opportunity to further enhance performance.

In [3], TSMC proposed a double-sided routing architecture, where a device layer is positioned between the front-side and back-side metal layers. Fig. 1 gives an example of the double-sided routing architecture, where three metal layers M_1, M_2, M_3 are on the front side and two metal layers BM_1, BM_2 are on the back side. The device layer facilitates signal connections between two sides using *bridging cells*. Any signal that needs to pass from the front side to the back side or in the reverse direction requires a bridging cell. A bridging cell occupies a certain area on the device layer, and has one front-side pin on M_1 and one back-side pin on BM_1. A bridging cell behaves like a buffer, and its pins are the input and output pins. Fig. 1 also shows three possible routing solutions for a net consisting of four front-side pins p_1, p_2, p_3, p_4 on M_1. Fig. 1(a) gives a routing solution whose wire segments are all connected on the front side. Fig. 1(b) is a routing solution with wire segments on both sides. For the latter solution, two of the four pins, p_1 and p_2, are respectively connected to the front-side pins of bridging cells b_1 and b_2 on the front side while the back-side pins of bridging cells b_1 and b_2 are connected on the back side. For a pin that needs a bridging cell to bring the signal to the other side, e.g., p_1 or p_2 in Fig. 1(b), we call it a *crossing pin*. Fig. 1(c) shows a routing solution that demotes the entire net to the back side and makes all pins become crossing pins.

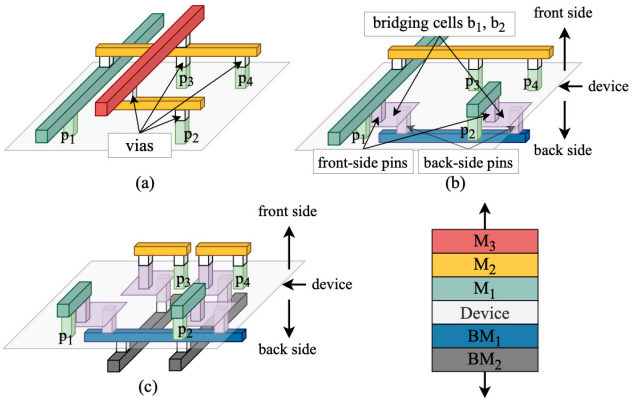


Fig. 1: Routing solutions of a net: (a) A single-sided solution on the front side. (b) A double-sided solution with some pins but not all pins being crossing pins. (c) A double-sided solution with all pins being crossing pins.

According to the example given in Fig. 1, a net can be routed as a *front-side net* (e.g., Fig. 1(a)) or a *double-sided net* (e.g., Fig. 1(b) and (c)). If a net is routed as a double-sided net, it is in fact transformed into individual nets, and each net is routed on a single side. For the routing solution in Fig. 1(b), the original 4-pin net is transformed into three nets: $\{p_1, p_3, p_4, b_1\_fp\}$ on the front side, $\{p_2, b_2\_fp\}$ on the front side, and $\{b_1\_bp, b_2\_bp\}$ on the back side, where b_i_fp

*This work was supported in part by the National Science and Technology Council under grant 111-2221-E-007-119-MY3.

and b_i_bp respectively denote the front-side and back-side pins of the bridging cell b_i of p_i. For the routing solution in Fig. 1(c), the original 4-pin net is transformed into five nets: $\{p_1, b_1\_fp\}$ on the front side, $\{p_2, b_2\_fp\}$ on the front side, $\{p_3, b_3\_fp\}$ on the front side, $\{p_4, b_4\_fp\}$ on the front side, and $\{b_1\_bp, b_2\_bp, b_3\_bp, b_4\_bp\}$ on the back side.

To our best knowledge, no existing academic or commercial routing tools are able to solve the double-sided routing problem. Nevertheless, if each net can be kept intact (as a front-side net) or transformed into individual nets (as a double-sided net) beforehand, existing routers can be leveraged to perform routing on each side separately after placement legalization to find legal locations for bridging cells and other cells is done. This motivates us to study a netlist planning problem for double-sided signal routing. We present an effective approach that is able to properly keep each net intact or transform it into individual nets under the consideration of routing congestion. Our netlist planning approach has been combined with a commercial place and route tool, and the experimental results shows that our approach effectively enables the commercial tool to produce high-quality double-sided routing solutions.

II. Problem Formulation

In this section, we formally define the netlist planning problem. We assume that (1) the PDN and clock network have already been constructed, (2) there are a set of m metal layers on the front side (denoted as $F = \{M_1, M_2, ..., M_m\}$) and a set of n metal layers on the back side (denoted as $B = \{BM_1, BM_2, ..., BM_n\}$), and (3) there are a set of preferred routing layers (denoted as $R = B \cup F'$ with $F' \subset F$).

We are given a placed netlist N containing two sets of signal nets, N_1 and N_2, where N_1 is the set of critical nets and N_2 is the set of non-critical nets. The netlist planning problem asks to transform the netlist N into a front-side netlist and a back-side netlist as well as to determine a rough location for each inserted bridging cell. Moreover, the addressed problem also expects that a high-quality double-sided routing solution can be produced when passing the planning result to a place and route tool under the requirement of routing critical nets in N_1 on the set R of preferred layers as much as possible.

III. Our Approach

A. Overall Flow

To solve the netlist planning problem, we formulate it as a 2D global routing problem on two 2D routing grid graphs, where each grid graph models the routing resources on one side. The flowchart shown in Fig. 2 outlines our approach, where the steps colored in white adopt similar ideas from an open-source global router NTHU-Route 2.0 [4] but work on two grid graphs. Our approach begins with the initial stage, where the metal layers on both sides are projected into two 2D grid graphs. Then the net partitioning step distributes the pins of each net to one or two 2D grid graphs. In the initial routing step, FLUTE [5] is first applied to decompose each multi-pin net into a set of two-pin nets on each 2D grid graph. For each two-pin net, L-shaped pattern routing and edge shifting

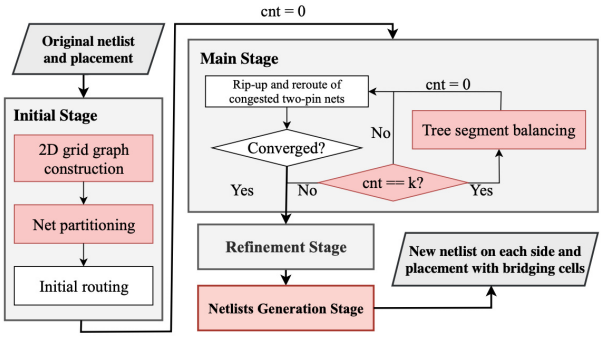


Fig. 2: Flowchart of our approach.

techniques are then employed to generate the initial routing from the corresponding 2D grid graph.

In the main stage, congested two-pin nets (i.e., nets passing through edges with overflow) on each side are iteratively ripped up and rerouted using monotonic routing or maze routing under a cost function considering congestion, wirelength and expected via count. After each iteration, the convergence of routing overflow on both sides is checked. If convergence is not achieved, the next iteration continues. After k iterations, a tree segment balancing step is performed to distribute two-pin nets more evenly between the front side and back side. The rip-up and reroute step then continues to further refine the routing solution on each side. In the refinement stage, it resembles the main stage, but the cost function is adapted to focus on finding overflow-free paths for congested two-pin nets. Finally, the netlist generation stage generates a new netlist on each side and the positions of bridging cells. In the rest of this section, we detail those steps with red color in Fig. 2.

B. 2D Grid Graph Construction

For our problem, we project the metal layers on both sides to two separate 2D grid graphs. In each grid graph, a vertex represents a GCell on the corresponding side, and two abutting GCells are connected by an edge. Each pin is positioned at a GCell, but when two or more pins of a net are located at the same GCell, they are grouped into a single pin. The routing resource estimation is based on an idea from CUGR [6]. The capacity of each edge in a grid graph indicates the number of available routing tracks between two adjacent GCells. The demand on an edge represents the number of nets passing through that particular edge. The overflow of an edge is the amount by which the demand surpasses the capacity, indicating potential congestion.

C. Net Partitioning

The net partitioning step consists of three phases: 1) initial partitioning, 2) pin redistribution, and 3) connection point assignment.

1) Initial partitioning: In the initial partitioning phase, our goal is to assign as many pins as possible to the back side, particularly for pins belonging to critical nets. This allows us to route more wire segments on the back side. When assigning a front-side pin to the back side (or assigning a back-side pin, such as an I/O pin, to the front side), we need to insert a

bridging cell on the device layer and therefore we also partition the device layer into an array of GCells. To effectively manage the number of inserted bridging cells we introduce the bridging cell capacity in a GCell on the device layer. The bridging cell capacity within a Gcell v is calculated using the following equation, which estimates the number of bridging cells that can be accommodated within the remaining space of the GCell.

$$bridging\_cell\_capacity(v) = \frac{GCell_{area} - cell_{area}(v)}{bridging\_cell\_area} \quad (1)$$

where $GCell_{area}$ is the area of a GCell, $cell_{area}(v)$ is the total area occupied by the cells and macros in the GCell v, and $bridging\_cell\_area$ is the area of a bridging cell.

Based on our empirical observations, if the amount of bridging cells inserted in a GCell exceeds its upper bound, there is a possibility that the excess bridging cells can be accommodated in neighboring GCells by displacing neighboring standard cells later during placement legalization. The feasibility of inserting a bridging cell at GCell v depends on whether there is sufficient bridging cell capacity in v or a neighboring GCell located within the distance r from GCell v.

The initial partitioning phase begins by prioritizing nets based on criticality and then sorts them based on their bounding box sizes. Next, it examines the positions of the pins of each net and determines which pins can be assigned to the back side, taking into account the bridging cell capacities. Additionally, we consider the potential drawback of having a significant overlap between the bounding boxes of a net's pins when assigning some pins to the other side, which may result in longer wirelength during subsequent routing stages. To mitigate this drawback, if it is feasible to demote all the pins of a net to the back side, we will assign all the pins to the back side in the initial partitioning phase. It is important to note that the net order ensures that nets with higher criticality or longer wirelength are given priority to use the bridging cells first.

2) Pin redistribution: After obtaining the initial partitioning result, we generate a corresponding RUDY map for each side based on this result. The RUDY map allows us to identify potential congestion regions on a side. The RUDY value contributed to GCell v by a net q is denoted by $RUDY_q(v)$ and represents the average wirelength per unit area within the bounding box of q [7]. It provides us with an estimation of the wire density and congestion likelihood for each net. The RUDY value of a GCell v is calculated by aggregating the RUDY values of all nets in the netlist M on a side as shown below.

$$RUDY(v) = \sum_{n \in M} RUDY_n(v) \quad (2)$$

We estimate the congestion of a GCell by adding the RUDY value and number of pins in the GCell. We construct separate congestion maps for the front side and back side. To account for the differences in available resources on two sides, we normalize each value in a map by dividing it by the sum of the routing capacities of the edges surrounding the associated GCell v using the following equation.

$$cong\_map(v) = \frac{RUDY(v) + pin\_num(v)}{routing\_capacity(v)} \quad (3)$$

A congestion value greater than 1 indicates an excessive potential demand, while values greater than 1 across a large region show that achieving zero overflow during the rip-up and reroute step may be challenging. We identify congested GCells on the back side and non-congested GCells with the same coordinates on the front side (or vice versa) based on a threshold T. By quantifying the congested value difference between the two GCells, we can determine the number of pins that need to be moved from the congested GCell to achieve a more balanced congestion distribution on each side. The number of pins that can be moved from the front-side GCell v_f is recorded in $F2B(v_f)$, while the number of pins that can be moved from the back-side GCell v_b is recorded in $B2F(v_b)$.

For example, let's consider a scenario where the congestion value on the back-side GCell v_b is 2, and the congestion value on the front-side GCell v_f with the same 2D coordinate is 0.5. The routing capacity of v_b is 3, and the routing capacity of v_f is 10. To bring the congestion value on the back side to 1, we need to move $(2-1) \times 3 = 3$ pins according to Equation (3). On the other hand, to ensure that the congestion value on the front side does not exceed 1, we can move at most $(1 - 0.5) \times 10 = 5$ pins. In this scenario, $B2F(v_b)$, which represents the number of pins to be swapped from the back side to the front side is set to 3.

Algorithm 1: Pin redistribution

```
   Input: netlist N, current S_front, S_back
   Output: updated S_front, S_back
1  for iter = 1, 2, 3 do
2      T ← T − (0.1 ∗ iter);
3      Initialize candidates = {};
4      get_congmap(S_front, S_back);
5      F2B, B2F ←
         get_move_num(cong_map_front, cong_map_back, T);
6      N_sort ← sort N by criticality and bounding box size;
7      for i = 0, 1, .., |N| − 1 do
8          q ← N_sort[i];
9          for pin ∈ get_pin_set(S_front, q) do
10             v ← GCell of pin;
11             if F2B(v) > 0 and is_valid_movement(v) then
12                 candidates.insert(pin);
13                 F2B(v) ← F2B(v) − 1;
14     for i = |N| − 1, ..., 1, 0 do
15         q ← N_sort[i];
16         for pin ∈ get_pin_set(S_back, q) do
17             v ← GCell of pin;
18             if B2F(v) > 0 and is_valid_movement(v) then
19                 candidates.insert(pin);
20                 B2F(v) ← B2F(v) − 1;
21     for pin ∈ candidates do
22         S_front, S_back ← move_pin(pin);
```

The pin redistribution phase will perform three iterations on gradually decreasing values of the threshold T using Algorithm 1. Before the algorithm starts, S_{front} and S_{back} respectively store the set of pins assigned to the front side and the set of pins assigned to the back side, which are produced by the initial partitioning phase. In lines 3-5, it initializes the

candidate pins to be moved and calculates the congestion maps for both sides. Line 6 sorts the nets in N. Lines 7-13 prioritizes critical nets and records the number of pins to be moved from the front side to the back side. Lines 14-20 prioritize non-critical nets and record the number of pins to be moved from the back side to the front side. Lines 21-22 update S_{front} and S_{back}. The net order helps increase the chances of critical nets utilizing back-side resources. Note that we check if pins can be moved using the function $is\_valid\_movement()$. This function verifies the need for a bridging cell and checks the availability of sufficient bridging cell capacity. The function $get\_pin\_set()$ returns the set of pins that are stored in S_{front} or S_{back} for a net.

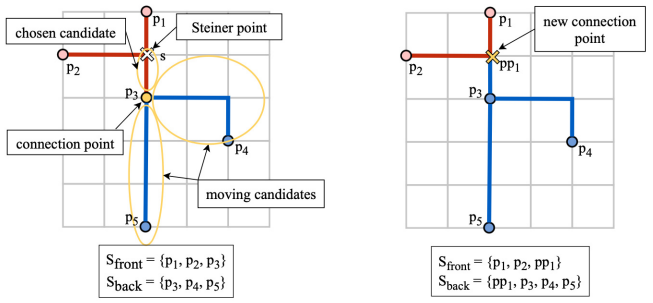
Initially, the threshold T is set higher to prioritize the resolution of highly congested regions. By focusing on these regions first, we can effectively address the most severe congestion areas. Subsequent iterations gradually reduce the threshold, allowing for the resolution of less congested regions.

3) Connection point assignment: After pin redistribution, a *connection point* is required to connect the signals between the front side and back side for a double-sided net q like the one shown in Fig. 1(b). (Note that for the routing in Fig. 1(c), the four individual nets on the front side will not be present during global routing as each pin and the front-side pin of its bridging cell are located at the same GCell.) To minimize the additional wirelength caused by connecting the front-side and back-side nets for net q, we consider the bounding boxes of the pins of q assigned to the front side and the back side to determine the connection point. If the bounding boxes have an intersection, a connection point is chosen within the intersection region. If no intersection exists, it extends both bounding boxes simultaneously until an intersection is found. Within the intersection region, as long as there is a crossing pin, choose one as the connection point arbitrarily. If no crossing pin is present, the algorithm selects a GCell v on the device layer with sufficient bridging cell capacity to place a bridging cell as the connection point. If a valid connection point cannot be found within the intersection, the intersection region is extended until a suitable connection point is identified.

D. Tree Segment Balancing

In the main stage and refinement stage, congested two-pin nets are ripped and rerouted on two separate sides. This means that the partitioning of pins remains fixed during the rerouting process. However, there may be regions where the routing capacities on one side are already saturated and cannot be improved by rerouting alone. To address this issue, the tree segment balancing step redistributes tree segments (i.e., routed two-pin nets) between two sides by swapping congested segments into the other side. This step provides an opportunity to change the pin distribution result produced from the net partitioning step.

1) Overview: As mentioned previously, each net in our approach is connected by at most one connection point, resulting in at most one subtree per side throughout the entire 2D global routing process. In the tree segment balancing step, only two-pin nets that do not disrupt the overall structure of their original multiple-pin net can be selected as candidates for segment swapping.



(a) Before moving. (b) After moving.

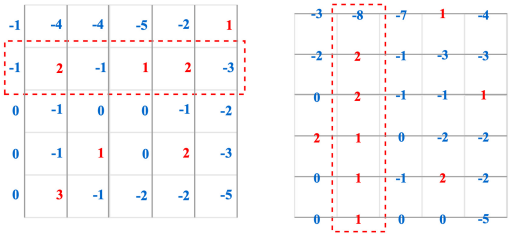
Fig. 3: An example of tree segment balancing.

Fig. 3(a) shows an example of a five-pin net that is decomposed into five two-pin nets. Among them, three red segments are located on the front side, and the other two blue segments are on the back side with the crossing pin p_3 as a connection point. We aim to select a two-pin net that offers the most improvement score among the movable candidates within the yellow circles. Once selected, the demand from the same path of this two-pin net is transferred to the grid graph on the other side, and the connection point is updated accordingly.

For a net, the pins assigned to the front side and the back side will change after the movement of a segment. In the example shown in Fig. 3(b), the new connection point is located at one endpoint of the candidate segment. As this new connection point serves as a Steiner point, a pseudo pin pp_1 is created and added to the pin sets on both sides. It is worth noting that the usage of bridging cells is taken into consideration during this step. Each segment swap requires an assessment of whether there are sufficient bridging cell capacities available in the GCell on the device layer.

2) Scores of candidates: The segment selection is crucial as it directly impacts the potential reduction of congestion in the updated routing tree. We adopt a routability difficulty score in [8] to assess the difficulty of resolving congestion in the bounding box of each two-pin net. To calculate the difficulty score, we assess the congestion level using the over-usage map within the bounding box of each two-pin net. Fig. 4 illustrates the distribution of over-usage values in both the row and column directions, with the most congested row and most congested column, having the highest sum of red numbers, highlighted in red boxes. In the figure, vertical/horizontal edges are categorized as either overflowed (with positive values) or slack (with negative values) within each row/column. The evaluation method assesses the routability of the most congested row/column by redistributing overflows from overflowed edges to the nearest slack edges, eliminating congestion.

The difficulty score is calculated by iteratively examining each vertical/horizontal edge, starting from the right/top and moving towards the left/bottom, as illustrated in Fig. 5. The



(a) Vertical direction. (b) Horizontal direction.

Fig. 4: Distribution of over-usage values for a two-pin net.

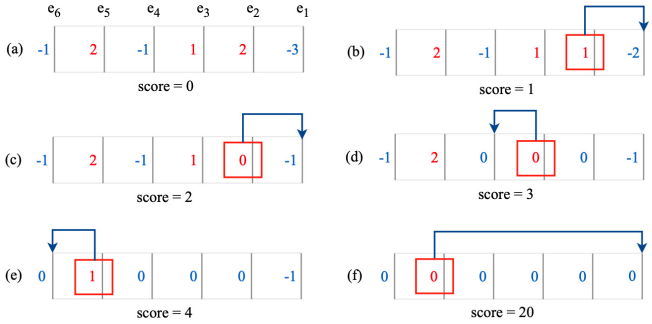


Fig. 5: An example of calculating the row difficulty score of a two-pin net.

score is determined by assigning the overflowed demand to the nearest slack edge and summing the square of its moving distance. In the provided example (Fig. 5(a)), let us consider edge e_2, which has two units of overflow. By moving one unit of overflow from e_2 to e_1 twice, the score increases by 2 (square of 1 + square of 1, as shown in Fig. 5(b) and (c)). Moving on to edge e_3, which has one unit of overflow, we shift it to e_4, resulting in an increase in the score by square of 1 (see Fig. 5(d)). This process continues for the remaining edges, and in the end, the difficulty score for this particular row is calculated to be 20 (see Fig. 5(e) and (f)).

The difficulty score of a two-pin net is obtained by summing up the difficulty scores of the most congested row and the most congested column. We determine the difficulty score of the two-pin net on the original side before the move and the difficulty score on the destination side assuming the demands of the candidate are transferred. The candidate with the highest improvement in score is chosen.

E. Congestion-aware Planning for Critical Nets

To prioritize the routing of critical nets on the preferred layers, we employ an idea from [9] to record virtual capacity and virtual demand on each edge additionally. The virtual capacity of an edge is determined by aggregating the capacities on its preferred routing layers, while the virtual demand takes into account the number of critical nets and the proportion of non-critical nets routed through the edge. When routing a critical net through an edge, the cost function is determined by the virtual demand and virtual capacity associated with that edge. This refined cost function allows us to better estimate the congested regions that critical nets are likely to encounter on their preferred routing layers.

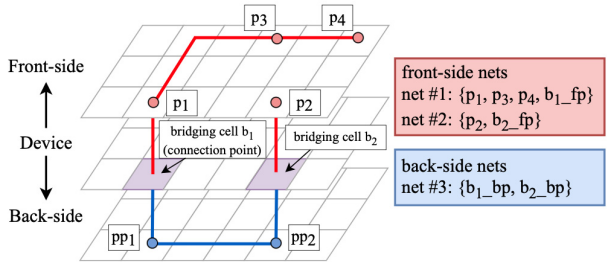


Fig. 6: An example of a netlist planning result.

F. Netlists Generation

This stage generates the netlist for each side and the location for each inserted bridging cell. Let us use Fig. 6 to explain this stage. Fig. 6 shows a global routing result on each side for a net with four pins p_1, p_2, p_3 and p_4. For this net, pin p_1 serves as both a crossing pin and the connection point. The first front-side net includes pins p_1, p_3, p_4, and the front-side pin of the bridging cell b_1, i.e., b_1_fp. The second front-side net has the crossing pin p_2 and the front-side pin of the bridging cell b_2, i.e., b_2_fp. The third net is a back-side one that connects the back-side pins b_1_bp and b_2_bp of the two bridging cells. In addition, each bridging cell is given a location that is at the center of the corresponding GCell on the device layer. Note that inserted bridging cells may overlap with other cells, which can be resolved by a placement legalizer.

IV. Experimental Results

A. Experimental Settings

Our approach was implemented in C++ and executed on a Linux machine equipped with an AMD EPYC 7282 16-Core 2.8 GHz CPU and 256GB of memory. In order to evaluate the performance of our approach, we modified the benchmarks from the ISPD'18 detailed routing contest [10]. However, we excluded the test case ispd18_test10 from our evaluation, as its placement density exceeds 90%, which is not suitable for our problem. The original ISPD'18 benchmarks consist of 9 metal layers. In our modifications, we added two thicker metal layers as back-side metal layers BM_1 and BM_2. For I/O pins, we assigned them to the bottom metal layer on the back side, i.e., BM_2. We took the size of an inverter cell to simulate the size of a bridging cell. Furthermore, we chose the nets whose bounding box sizes are among top 3% as critical nets. The metal layers M_8 and M_9 on the front side as well as the metal layers BM_1 and BM_2 on the back side were chosen as preferred routing layers for critical nets. That is, the top 2 layers on the front side are the preferred routing layers for a front-side net which is part of a critical net, while all layers on the back side are the preferred routing layers for a back-side net which is part of a critical net. To simulate the impact of the PDN, we reduced the routing capacities on the back side by 50%.

For comparison, we also made modifications to the ISPD'18 benchmarks to generate a single-sided routing architecture without back side. We added two thicker metal layers as the top two layers, i.e., M_{10} and M_{11}. We moved the I/O pins to

TABLE I: Comparison between double-sided routing results and single-sided routing results.

Case name	Benchmark information			Single-sided routing				Double-sided routing				CPU (s)
	#Nets	#Pins	Placement density (%)	Wirelength	Vias	Percentage of CN (%)	DRCs	Wirelength	Vias	Percentage of CN(%)	DRCs	
ispd18_test1	3153	17203	85.21	103853	35526	63.385	0	91763	39519	64.194	0	0.20
ispd18_test2	36834	159201	47.92	1687365	333204	90.486	3	1679857	382675	93.355	0	5.98
ispd18_test3	36700	159703	57.32	1921023	321074	96.134	57	1809381	368439	96.682	0	7.43
ispd18_test4	72410	318245	89.05	3400676	814609	71.459	19	2868166	784907	81.338	2	35.87
ispd18_test5	72394	318195	81.76	3579993	1004767	78.518	45	2948983	927332	86.477	0	71.27
ispd18_test6	107701	475541	84.38	4069433	1505562	55.383	4	3742979	1444838	67.911	1	17.56
ispd18_test7	179863	793289	82.96	7651626	2555298	67.183	67	7258221	2255497	83.272	0	545.00
ispd18_test8	179863	793289	83.12	7717104	2563281	67.444	67	7174484	2259641	83.173	1	483.63
ispd18_test9	178858	791761	86.22	6215012	2594732	57.558	393	5654361	2273769	73.381	0	85.82
Comp.	-	-	-	1.000	1.000	1.000	1.000	0.909	0.982	1.136	0.006	-

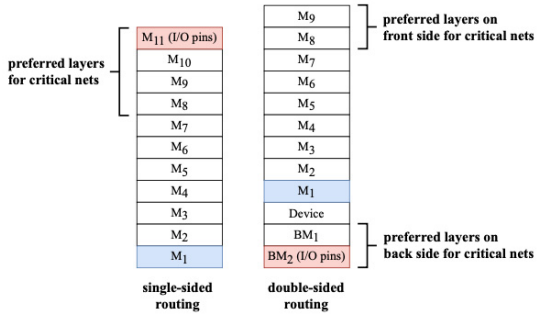


Fig. 7: Two configurations in the experiment.

M_{11} and assumed the PDN to occupy the top 2 layers, leaving the remaining half of the routing capacities for signal routing. Additionally, we introduced dropped vias on lower metal layers to connect the power nets to the standard cells. The top four metal layers, i.e., M_8, M_9, M_{10}, M_{11}, were chosen as preferred routing layers for critical nets. The single-sided routing architecture and double-sided routing architecture are illustrated in Fig. 7.

B. Comparison between Double-sided Routing and Single-sided Routing

To evaluate our approach, we used a commercial place and route tool, Cadence Innovus 21.13 [11], to produce a double-sided routing solution for each benchmark. The commercial tool took each netlist planning result produced by our approach as the input and performed placement legalization (to find legal locations for all cells including bridging cells) and routing on each side separately (to get a double-sided routing solution). For critical nets, we asked the commercial tool to route them on the preferred layers as much as possible. We also used Innovus to perform routing directly on each benchmark to produce the single-sided routing solution.

Table I gives the information of each benchmark and the single-sided and double-sided routing results. For critical nets, the "percentage of CN (%)" column of Table I gives the ratio of the total length of their wire segments routed on the preferred layers to their total wirelength. The "CPU" column gives the runtime taken by our netlist planning approach. Compared to single-sided routing, double-sided routing not only improved the wirelength by 9.1% and the via count by 1.8% but also achieved zero or near zero DRC violations. Double-sided routing also increased the percentage of wirelength of critical nets routed on the preferred routing layers by 13.6%. These encouraging results indicate that our netlist planning approach is effective in the sense that it enabled the commercial tool to produce high-quality double-sided routing solutions. These results also demonstrate the potential benefit of performing double-sided routing for signal nets. In addition, our approach took from less than 1 second to about 9 minutes to produce a netlist planning result, which is quite acceptable considering the problem size of each benchmark in our experiment.

V. Conclusion

We formulated a netlist planning problem for double-sided signal routing and presented an effective approach to enable the generation of high-quality routing solutions by a commercial place and route tool. To our best knowledge, this is the first work that addresses double-sided routing for signal nets.

References

[1] E. Beyne, A. Jourdain, G. Beyer, and G. van der Plas, "Nano-through silicon vias (ntsv) for backside power delivery networks (bspdn)," in *Proceedings of Symposium on VLSI Technology and Circuits*, 2023.

[2] Agnihotri, M. Asoro, M. Aykol, B. Bains, R. Bambery, M. Bapna, A. Barik, A. Chatterjee, P. C. Chiu, T. Chu, *et al.*, "Intel powervia technology: Backside power delivery for high density and high-performance computing," in *Proceedings of Symposium on VLSI Technology and Circuits*, 2023.

[3] S.-H. Chen, J. C. J. Kao, K.-N. Yang, and J. Liu, "System and method for back side signal routing," *U.S. Patent 11423204*, 2022.

[4] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang, "Nthuroute 2.0: a robust global router for modern designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 12, pp. 1931–1944, 2010.

[5] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2007.

[6] J. Liu, C.-W. Pui, F. Wang, and E. F. Young, "Cugr: Detailed-routability-driven 3d global routing with probabilistic resource model," in *Proceedings of Design Automation Conference*, 2020.

[7] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Proceedings of Design, Automation and Test in Europe*, 2007.

[8] W.-H. Liu, T.-K. Chien, and T.-C. Wang, "Metal layer planning for silicon interposers with consideration of routability and manufacturing cost," in *Proceedings of Design, Automation and Test in Europe*, 2014.

[9] T.-H. Lee, Y.-J. Chang, and T.-C. Wang, "An enhanced global router with consideration of general layer directives," in *Proceedings of International Symposium on Physical Design*, pp. 53–60, 2011.

[10] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H. Liu, "Ispd 2018 initial detailed routing contest and benchmarks," in *Proceedings of International Symposium on Physical Design*, pp. 140–143, 2018.

[11] *Innovus Implementation System.*, 21.13 ed. https://www.cadence.com/.