ICCAD 2018 CAD Contest

Obstacle-Aware On-Track Bus Routing

About Liao, Hua-Yu Chang, Owen Chi, and Jane Wang
Synopsys Taiwan Co., Ltd.,
1F., No. 25, Gongye E. 4th Rd., Hsinchu Science Park, Hsinchu, Taiwan

Contents

0. Announcements

- FAQ updated- 2018/4/12
- FAQ updated- 2018/4/25

ICCAD 2018 CAD Contest

Obstacle-Aware On-Track Bus Routing

About Liao, Hua-Yu Chang, Owen Chi, and Jane Wang
Synopsys Taiwan Co., Ltd.,
1F., No. 25, Gongye E. 4th Rd., Hsinchu Science Park, Hsinchu, Taiwan

I. Introduction

Bus routing in advanced technology node is a challenging task due to: a) non-uniform and complex routing track configuration; and b) need to route in between small obstacles while maintaining the same routing topology for all bus bits.

Routing tracks (tracks) are essential in advanced technology node to help router adhere to design rules and help mask coloring. In order to meet the routing requirements for different buses (e.g. different wire width), uniform track configuration is not sufficient. The tracks in this problem have width constraint which only allows wires with width smaller than or equal to the constraint to route on them, and may not fully cover the whole design. Figure 1 illustrates a simple configuration of three types of tracks. The green tracks have the smallest width constraint, blue tracks have the larger width constraint, and red tracks have the largest width constraint. Also, the green and the red routing tracks cover the whole design from top to bottom, whereas the blue routing tracks only cover partial of the design.

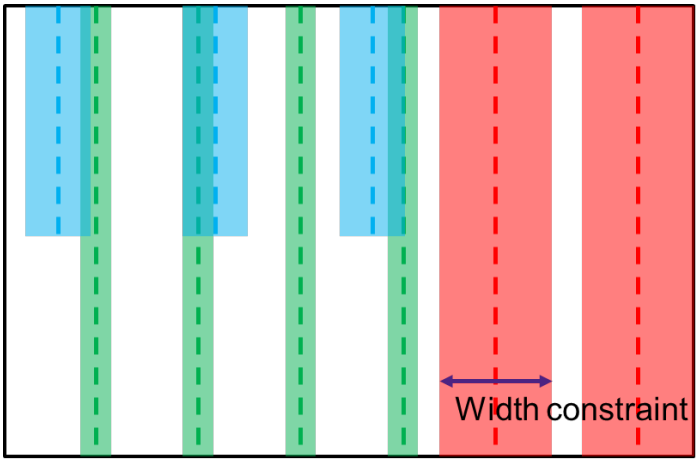

Figure 1 Routing Track Example

Small obstacles such as power vias make bus routing even more challenging. Since such obstacles are scattered throughout certain layers, it is not possible to find continuous routable area if bus bits are not allowed to route in between some of them. Figure 2 shows an example of routing a bus on track through small obstacles while

maintaining same routing topology for all bus bits. The dotted lines are tracks, the red rectangles are obstacles, the orange rectangles with bolded outlines are pins, and the orange thick lines are routed paths.

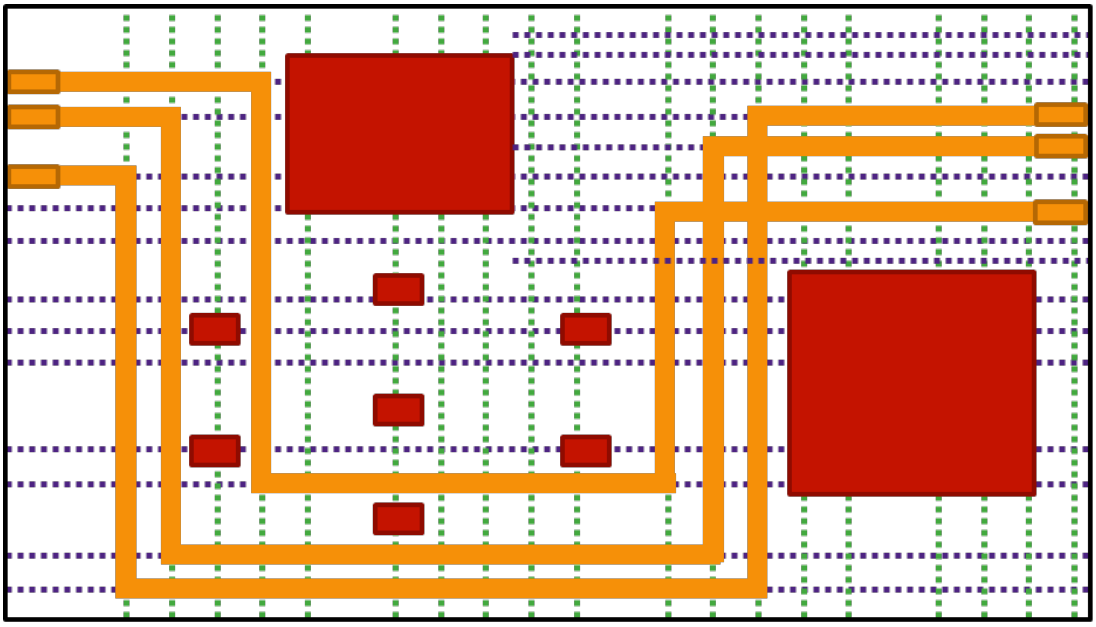

Figure 2 Bus Routing Example

II. Problem Formulation

Given the following inputs:

- A design boundary $\boldsymbol{BD}$.
- A set of layers $\boldsymbol{L} = \{L_1, L_2, ..., L_k\}$, where layer L_i, $1 \leqq i \leqq k$, defines
  - Routing direction $\boldsymbol{D_i}$.
  - Spacing $\boldsymbol{S_i}$.
- A set of tracks $\boldsymbol{T} = \{T_1, T_2, ..., T_m\}$, where track T_i, $1 \leqq i \leqq m$, defines
  - Track width $\boldsymbol{WT_i}$.
- A set of buses $\boldsymbol{B} = \{B_1, B_2, ..., B_n\}$, where bus B_i, $1 \leqq i \leqq n$, defines
  - Number of bits $\boldsymbol{NB_i}$.
  - Number of pin shapes for each bit $\boldsymbol{NP_i}$.
  - Pin shapes for bit j, $1 \leqq j \leqq NP_i$, $\boldsymbol{P_{ij}} = \{P_{ij1}, P_{ij2}, ..., P_{ijNPi}\}$.
  - Bus width for each layer $\boldsymbol{WB_i} = \{WB_{i1}, WB_{i2}, ..., WB_{ik}\}$.
- A set of obstacles $\boldsymbol{O} = \{O_1, O_2, ..., O_u\}$.

The program should output a set of routing paths $p = \{p_1, p_2, ..., p_v\}$ such that p connects all pin shapes of all buses. Each path can be either a horizontal wire, a vertical wire, or a via connecting two layers.

Take Figure 2 for example, the input can be broken down into two layers as shown in Figure 3. The input for this design is as follows:

- 2 layers L_1 and L_2.

- 14 tracks on L_1 (green dotted lines) and 17 tracks on L_2 (blue dotted lines).
- 1 set of buses with 3 bits and 2 pin shapes for each bit on L_2 (bolded outline orange rectangles).
- 8 obstacles on L_1 and 9 obstacles on L_2 (red rectangles).

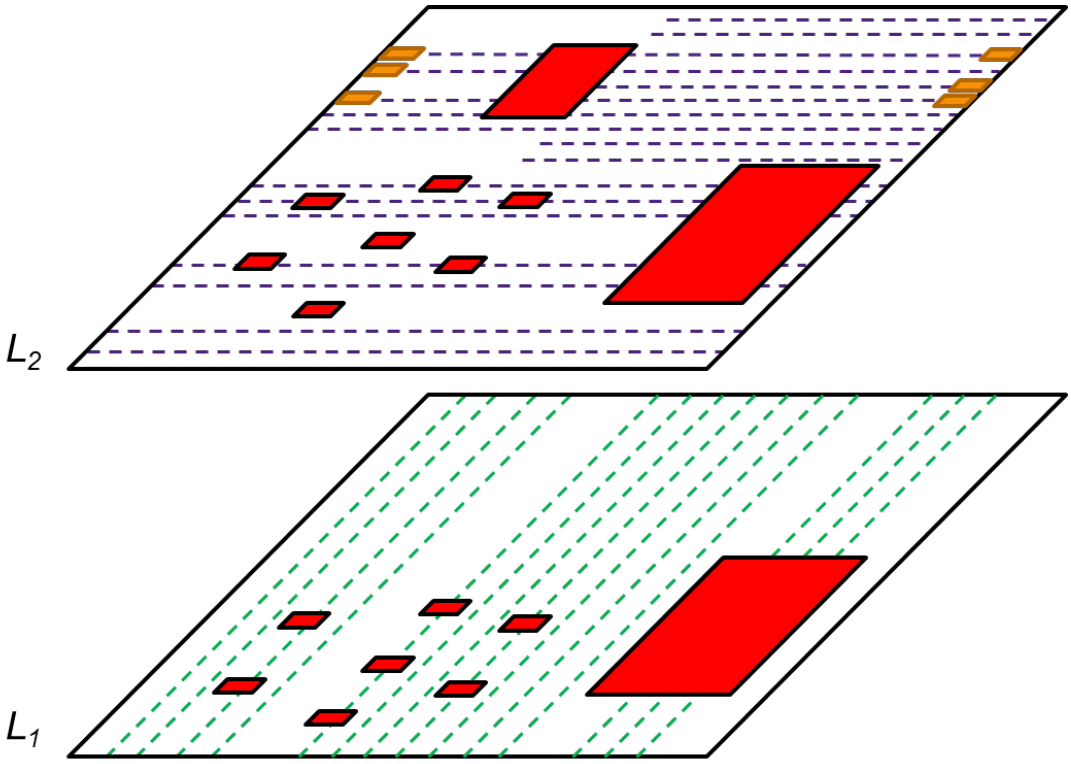

Figure 3 Input Example

The output of Figure 2 can be broken down into two layers as shown in Figure 4. The output is composed of 9 horizontal wires (thick orange rectangles) on L_2, 6 vertical wires on L_1, and 12 vias (thin orange lines).

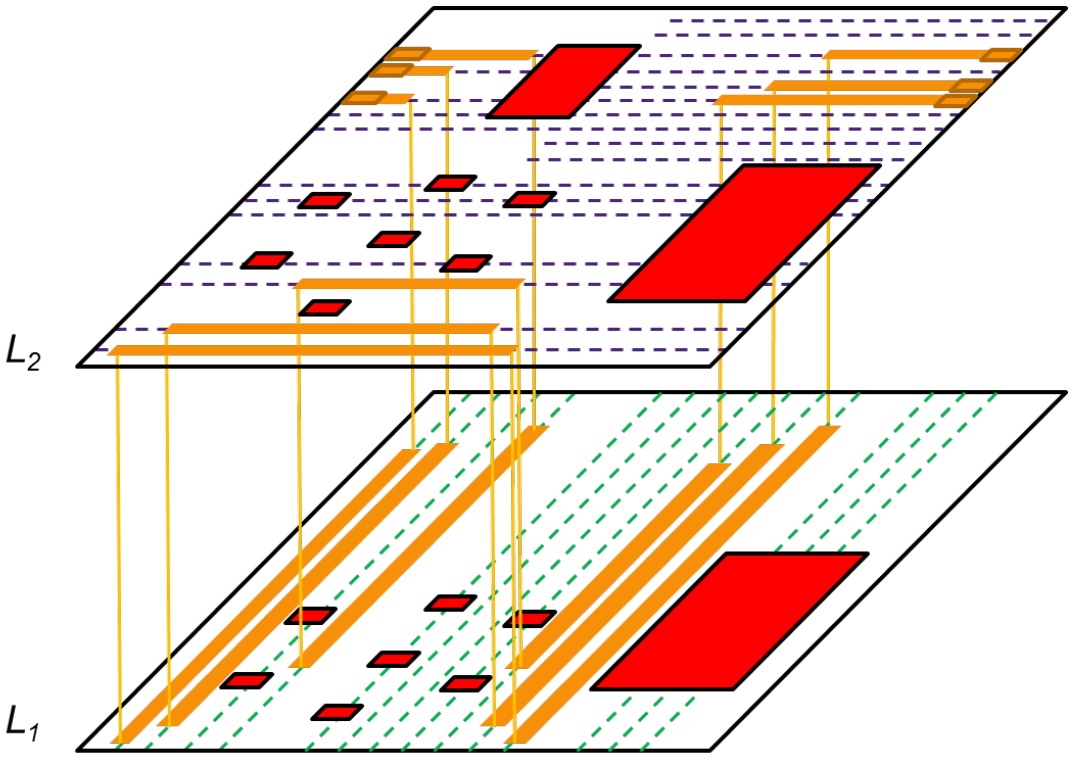

Figure 4 Output Example

III. Input Format Rules

A. Coordinates

The input objects, such as design boundary, pin shapes, tracks, obstacles, etc., are represented by either a point, a line, or a rectangle. The coordinates of the points, lines, and rectangles have the following rules:

- Coordinates are non-negative integers and are smaller than UINT32_MAX.
- A point is expressed by X coordinate followed by Y coordinate enclosed by parenthesis. E.g. (100 500).
- A line is expressed by two points with either the X coordinates or the Y coordinates being the same. E.g. a vertical line (100 500) (100 1000).
- A rectangle is expressed by two points with the first point being the lower left corner of the rectangle and the second point being the upper right corner of the rectangle. E.g. (100 500) (1000 1000).

B. Design boundary

Design boundary is represented by a rectangle. All other input objects are guaranteed to be enclosed by the design boundary. Input objects can touch the design boundary. E.g. design boundary (0 0) (1000 1000) and obstacle (0 100) (50 500). The design boundary has the following input format:

```
DESIGN_BOUNDARY <rectangle>
```

For example:

```
DESIGN_BOUNDARY (0 0) (1000 1000)
```

C. Layers

The layers are stacked from bottom to top based on their input order. They can be connected with neighboring ones through vias. For example, if the layers are L_1, L_2, and L_3, this means L_1 is the bottommost layer, and L_3 is the topmost layer. L_1 can connect to L_2, L_2 can connect to L_1 and L_3, and L_3 can connect to L_2.

Each layer has a routing direction and a spacing constraint. The routing direction is either horizontal or vertical. The tracks or the output routing paths on that layer must follow its routing direction. Note that neighboring layers do *NOT* guarantee to have different routing directions. The spacing constraint specifies the distance to which the output routing paths must keep with the obstacles, the design boundary, and other routing wires on that layer.

The layers have the following input format:

```
LAYERS <number of layers>
    [<layer name> <direction> <spacing>]...
ENDLAYERS
```

For example:

```
LAYERS 3
L1 horizontal 50
L2 vertical 100
L3 horizontal 80
ENDLAYERS
```

D. Tracks

The track is represented by a line on a layer with a width constraint. As mentioned in the previous section, the direction of the track is always the same as the routing direction of the layer that the track is on. For example, if L_1's routing direction is horizontal, all tracks on L_1 are horizontal.

Each track also comes with a width constraint. The width constraint represents the maximum width that a wire can route on. For example, suppose the width constraint of track T_1 is on L_1 and it width constraint is WT_1, and bus B_1's width constraint on L_1 is WB_{11}, then bus B_1 can only route on T_1 if WB_{11} is smaller than or equal to WT_1. The value of the width constraint is guaranteed to be multiples of 2.

Note that tracks can overlap with each other while having different width constraints. In this case, the output path can route on the track as long its width constraint is smaller than or equal to the maximum width constraints of the overlapping tracks. Take Figure 5 for example, if T_1 is (50 100) (500 100) on L_1 with width constraint $WT_1 = 10$,

and T_2 is (250 100) (1000 100) on L_1 with width constraint WT_2 = 20. In the overlapped segment of the two tracks, (250 100) (500 100), any routing path with width constraint smaller than or equal to 20 can route on it.

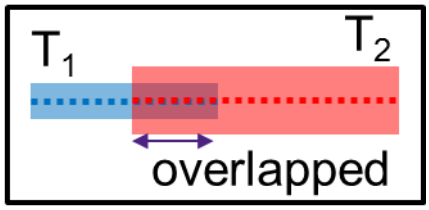

Figure 5 Track Overlap Example

The tracks have the following input format:

```
TRACKS <number of tracks>
    [<layer name> <line> <width constraint>]...
ENDTRACKS
```

For example:

```
TRACKS 4
L1 (0 10) (100 10) 50
L1 (0 20) (100 20) 50
L2 (10 0) (10 100) 100
L2 (20 0) (20 100) 100
ENDTRACKS
```

E. Buses

The bus is represented by the pin shapes of the bits and the width constraint on each layer. The bus has the following rules:

- Each bus specifies its number of bits and number of pin shapes for each bit. For example, for bus B_i, if its number of pin shapes is NP_i, then all bits have NP_i pin shapes.
- The number of bits is at least 1. Note that the number of bits can be 1. In this case, the bus acts like normal signal net.
- The number of pins shapes for each bit is at least 2.
- Each pin shape is represented by a rectangle on a layer. Pin shapes are guaranteed not to overlap with each other.
- The width constraint of the bus on a layer means the output routing paths for the bus on that layer has the width equal to the width constraint. The values of the width constraints of the bus are guaranteed to be multiples of 2.
- Each pin shape is guaranteed to overlap with at least one track which the width constraint of the track is larger than or equal to the bus's width

constraint on that layer. Note that pins may not align with the tracks perfectly. In real life, router needs to tap-off misaligned and off-track pins to nearest track. In this problem, we focus on routing the trunk of the bus, there will be no off-track pins. Also, for misaligned pins, output wires touching the pin shape is considered as connected as shown in Figure 6.

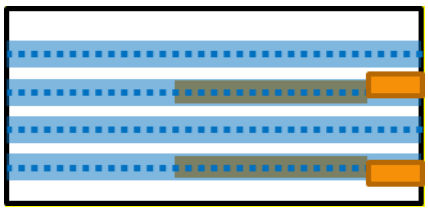

Figure 6 Misaligned Pin Example

The overlapped tracks might not be on the same layer as the pin shape. Take Figure 7 for example, there are 2 horizontal pin shapes on L_1, however, the tracks on L_1 (red tracks) are vertical. If routing started from L_1, the wires of both bits would overlap with each other (using the same red track). In this case, vias can first be placed in the pin shapes and start the routing from L_2 using different blue tracks.

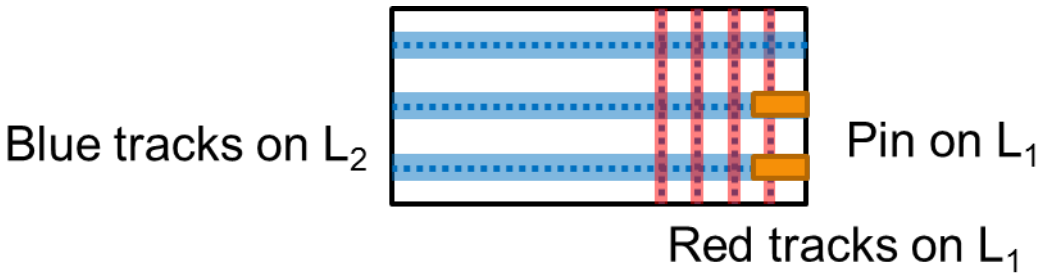

Figure 7 Pin Overlapped Tracks on Different Layers

- The order of the physical locations of the pin shapes of the bits is guaranteed to be either the same as that of the input bit name or the reverse of it (depends on how it is viewed). Take Figure 8 for example, bus B_1 has 3 bits, *b1<0>*, *b1<1>*, and *b1<2>*, and has 3 pin shapes for each bit on the left, bottom, and right of the design. Take the pin shapes on the left for example, their order from top to bottom is *b1<0>*, *b1<1>*, *b1<2>*, which is the same as the input order. (if viewed from bottom to top, it is of the reverse order of the input). For pin shapes on the bottom and right, if viewed from left to right and top to bottom, they are of the reverse order of the input. The pin shapes are guaranteed to be clustered into groups so that their ordering can be

distinguished clearly.

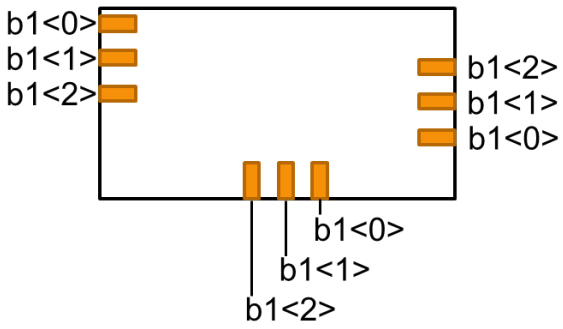

Figure 8 Pin Shape Location Order Example

The buses have the following input format:

```
BUSES <number of buses>
    [BUS <bus name>
    <number of bits>
    <number of pin shapes for each bit>
    WIDTH <number of layers>
        <width constraint>...
    ENDWIDTH
    [BIT <bit name>
        [<layer name> <rectangle>]...
    ENDBIT]...
    ENDBUS]...
ENDBUSES
```

For example:

```
BUSES 1
BUS B1
2
2
WIDTH 3
50
100
80
ENDWIDTH
BIT 0
L1 (0 0) (10 10)
L2 (100 200) (110 210)
ENDBIT
```

```
BIT 1
L1 (0 20) (10 30)
L2 (100 220) (110 230)
ENDBIT
ENDBUS
ENDBUSES
```

F. Obstacles

The obstacle is represented by a rectangle on a layer. Obstacles may overlap with each other.

The obstacles have the following input format:

```
OBSTACLES <number of obstacles>
    [<layer name> <rectangle>]...
ENDOBSTACLES
```

For example:

```
OBSTACLES 3
L1 (40 50) (60 100)
L1 (45 80) (100 200)
L2 (100 200) (120) (230)
ENDOBSTACLES
```

IV. Output Format Rules

The routing paths are written out to a file bus bit by bus bit. Each routing path can be either a horizontal wire on a layer, a vertical wire on a layer, or a via from a layer to a neighboring layer. The output of routing path has the following rules:

- If the routing path is a wire on a layer, it should be output in the format of the name of layer followed by the coordinates of the line. E.g. L1 (20 30) (50 30). The direction of the line should match the direction of the layer. Though the wire is represented by a line, it actually represents a rectangle shape that has width of the bus width constraint on that layer. Take Figure 9 for example, suppose p_1 is a horizontal wire, (20 30) (50 30), for bit $b1<0>$ of bus B_1 on L_1, the actual shape it represents is expanding the line by half the bus B_1 width constraint on L_1, which is $WB_1/2$, on both sides of the line like the figure on the right. Note that the rectangle the wire represents is used to check the spacing constraint with other objects (other wires, obstacles, etc.), not the line itself.

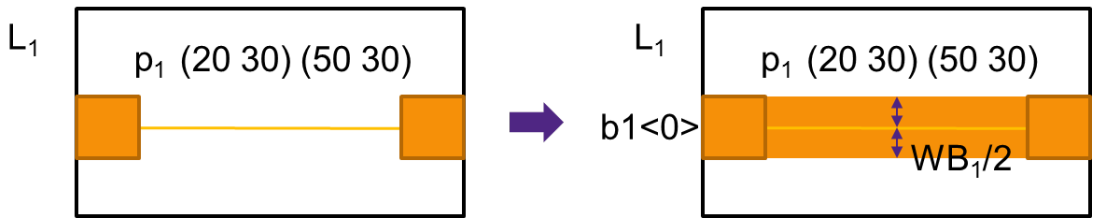

Figure 9 Line to Wire Example

- If the routing path is a via, it should be output in the format of a point and the lower layer of the two neighboring layers that it connects. E.g. L_1 (10 10) represents a via at (10 10) connecting L_1 to L_2. Outputting a via on the topmost layer is illegal.
- Routing paths of the same bit are considered to be connected if they overlap with each other. Take Figure 10 for example, there are 2 horizontal lines, p_1 and p_4, on L_1; 1 via, p_2, from L_1 to L_2, and 1 vertical line, p_3, on L_2. Suppose all of them belong to the same bit, then p_1, p_2, and p_3 are connected, whereas p_4 are not connected with the other three routing paths. Pin shapes touching or overlapping with routing paths are considered as connected. Stacking vias are allowed and are considered as connected. For example, suppose via 1 is at L1 (10 10), and via 2 is at L2 (10 10), then L1 is connected to L3 at point (10 10).

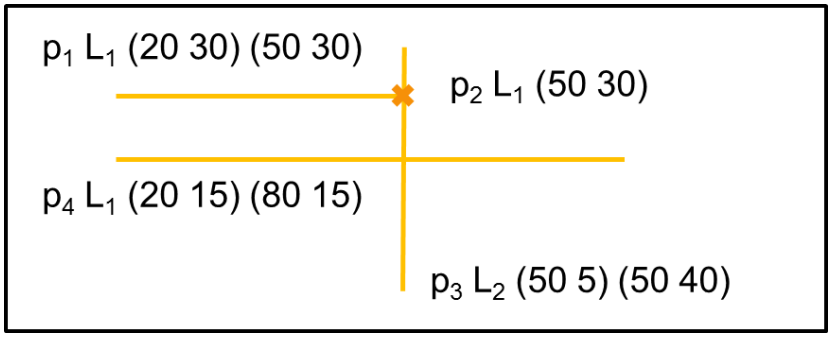

Figure 10 Connectivity Example

- The output order of the routing paths of a bus bit is irrelevant to results. That means, take Figure 10 for example, output order p_1, p_2, p_3, p_4, and p_4, p_3, p_2, p_1 represent the exact same results

The output has the following format:

```
[BUS <bus name>
    [BIT <bit name>
    PATH <number of routing paths>
```

```
        [<layer> <point> | <layer> <rectangle>]...
    ENDPATH
    ENDBIT]...
ENDBUS]...
```

For example:

```
BUS B1
BIT 0
PATH 3
L1 (20 30) (50 30)
L1 (50 30)
L2 (50 5) (50 40)
ENDPATH
ENDBIT
ENDBUS
```

V. Evaluation Methodology

The goal of the program is to complete the bus routing while minimizing the overall cost for each test case. The overall cost for a test case is defined as:

Overall_cost $\boldsymbol{C}$ = routing_cost $\boldsymbol{C_R}$ + penalty_cost $\boldsymbol{C_P}$

The routing cost is to measure the routing quality, such as routing resource used, bus topology, etc., and the penalty cost is to apply penalty to route fail buses and spacing violations between output routing paths. The routing cost only apply to buses that are routed successfully. The route fail buses will induce the penalty cost that is almost always higher than the routing cost if the bus is routed successfully. The contestants will be ranked by the **summation of all overall costs over all test cases**.

A. Route Success and Route Fail

A successfully routed bus has the following 3 requirements:

- Routing paths of a bit connect all pin shapes of the bit.
- All wires are on-track and do not violate the width constraint of the track.
- All bits are routed in the same topology.

The first requirement is straightforward, Figure 11 is an example of an unconnected bus.

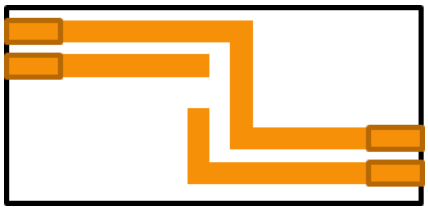

Figure 11 Unconnected Bus Example

For the second requirement, any wire off-track or has bus width constraint larger than the track width constraint is considered as route fail. Figure 12 shows both situations (note that horizontal tracks are not drawn). The one on the left has a vertical wire not on the vertical track. The one on the right has a vertical wire with width larger than the track width. Note that all tracks are within the design boundary, thus, any routing path outside of design boundary is definitely off-track and results in route fail of the bus.

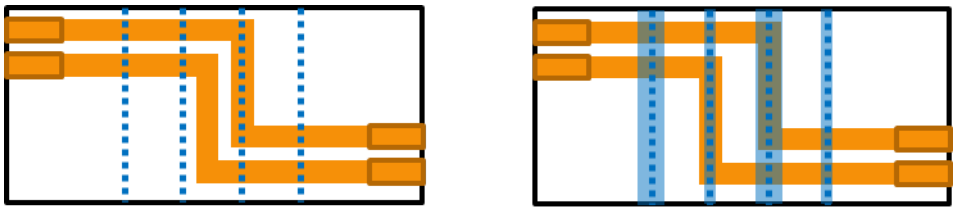

Figure 12 Track Violation Example

The third requirement is more complex to define. Here we break down routed bus into segments, where each segment represents a set of wires of different bits that have the same sequence when traced from a set of pin shapes. Take Figure 13 for example, suppose we start tracing wires from the set of pin shapes on the left, the two immediate horizontal wires (colored blue in the middle figure) connected to the pin shapes form the first segment Seg_1. Now, continue tracing from Seg_1 to the next set of wires, which are the vertical ones that form Seg_2 (the rightmost figure). In this example, this bus has 4 segments.

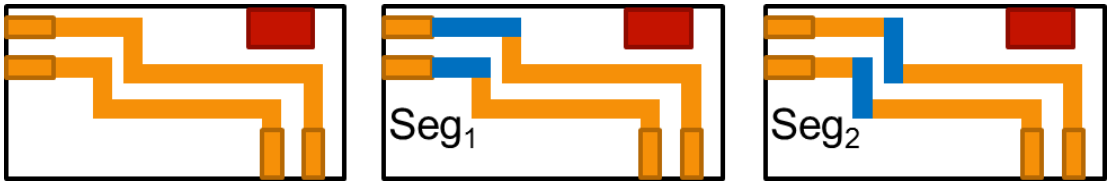

Figure 13 Bus Segment Example

Upon tracing the segments of the bus, the bits are considered as the same topology if the following four criterions are met:

- All bits should have the same number of wires.
- All wires traced from all bits should have the same layer sequencing. Take Figure 13 for example, suppose horizontal wires are on L_1 and vertical wires are on L_2 and suppose we start tracing the wires from the pin shapes on the left. Wires for both bits have the same layer sequencing: L_1, L_2, L_1, L_2.
- All wires traced from all bits should route towards the same direction. Take Figure 14 for example, the blue segment in the left figure has one bit routing

upward while the other one routing downward. Thus the bits are not considered as same topology. Note that segments can be T-junctions as well. The blue segment in the right figure is a T-junction. So, if one bit in a segment is a T-junction, all bits should to be T-junctions.

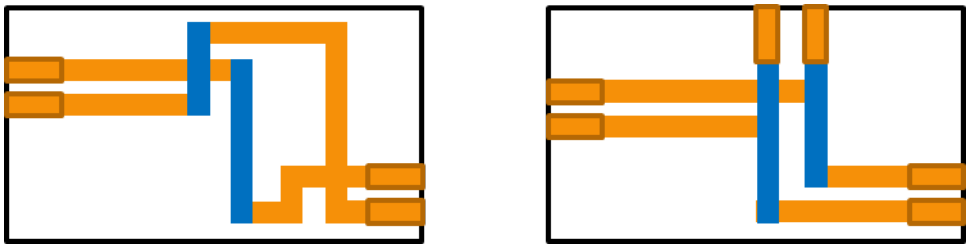

Figure 14 Traced Wire Direction Example

- Within each segment, the wires of different bits should maintain the same or the reverse order as the order seen from the pin shapes (as discussed in Section III.E). Two examples are given in Figure 15 to show bits routed in different topology and the bus is considered as route fail.

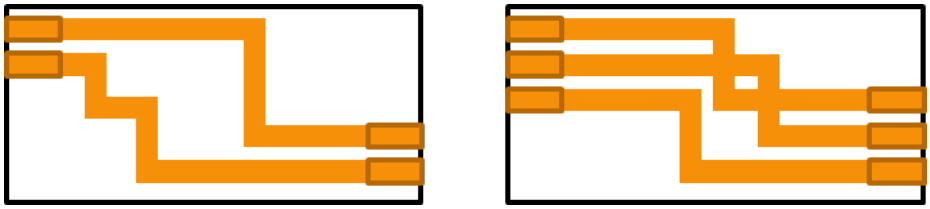

Figure 15 Different Topology Example

B. Routing Cost

If the bus is routed successfully, its routing cost is calculated. The routing cost $\boldsymbol{C_R}$ consists of three factors related to routing quality: the wire length cost $\boldsymbol{C_w}$, the segment cost $\boldsymbol{C_s}$, and the compactness cost $\boldsymbol{C_c}$. It is defined as:

$$\boldsymbol{C_R} = \sum_i^{All\ buses} \alpha \cdot \boldsymbol{C_{wi}} + \beta \cdot \boldsymbol{C_{si}} + \gamma \cdot \boldsymbol{C_{ci}}, \text{ where}$$

$$\boldsymbol{C_{wi}} = \frac{\sum_j^{All\ bits\ of\ bus\ i} \frac{wire\ length\ of\ bit\ j}{half\ parameter\ wire\ length\ of\ bit\ j}}{\#bits\ of\ bus\ i}$$

$$\boldsymbol{C_{si}} = \frac{\#segments\ of\ bus\ i}{lower\ bound\ of\ \#segments\ of\ bus\ i}$$

$$\boldsymbol{C_{ci}} = \frac{\sum_j^{All\ segments\ of\ bus\ i} \frac{width\ of\ segment\ j}{lower\ bound\ width\ of\ segment\ j}}{\#segments\ of\ bus\ i}$$

The wire length of a bit is calculated by summing up the length of all wires of that bit. The wire length cost for the bit is a normalized cost calculated by dividing the summed up length by the half parameter wire length of the bit. The wire length cost of a bus is the sum of the wire length cost of all bits divided by the number of bits. Thus, if

the bus is routed with minimum length required, the wire length cost of the bus should be close to 1. Take Figure 16 for example, taking detours will cause the wire length cost go up.

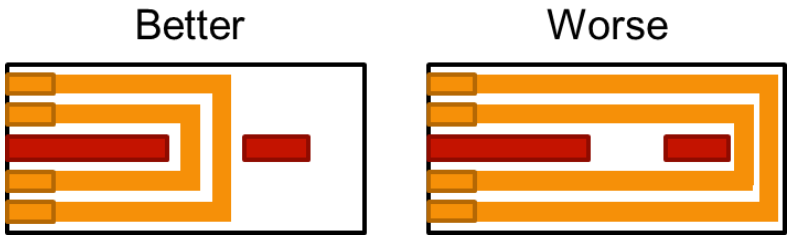

Figure 16 Wire Length Cost Example

The segment cost of a bus is defined as the number of segments of the bus divided by the lower bound number of segments of the bus. The lower bound is determined by the evaluation program based on the layer of the pin shapes and the location of the obstacles, to estimate how many segments are needed to route the bus. If the bus is routed with minimum number of segments, the segment cost of the bus should be close to 1.

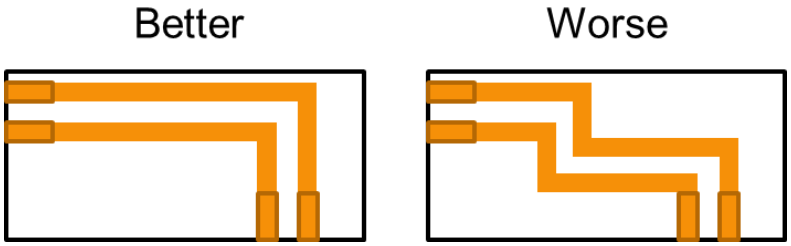

Figure 17 Segment Cost Example

The compactness cost of a segment is defined as the width of the segment divided by the lower bound width of the segment. If the direction of the segment is horizontal, the width of the segment is defined as the Y coordinate of topmost wire in the segment minus the Y coordinate of the bottommost wire in the segment. If the direction of the segment is vertical, the calculation is between the X coordinate of the rightmost and the leftmost wire.

The lower bound width of the segment is minimum width needed for placing all wires on that layer as compact as possible (without considering the tracks). For example, if S_1 is 50, NB_1 is 3, and WB_1 is 30, then the lower bound width of the segment on L_1 for B_1 is 160 since the minimum distance between wires is 80 (spacing plus width) and there are 3 bits. An exception is the segment connected to the pin shapes. The lower bound width for such segments is the distance between the topmost and bottommost pin shapes if the direction of the layer is horizontal, or is the distance between the

rightmost and leftmost pin shapes if the direction of the layer is vertical.

The compactness cost for a bus is the summation of the compactness cost of all segments divided by the number of the segments. If all segments are routed with widths close to the lower bound, the compactness cost for the bus should be close to 1.

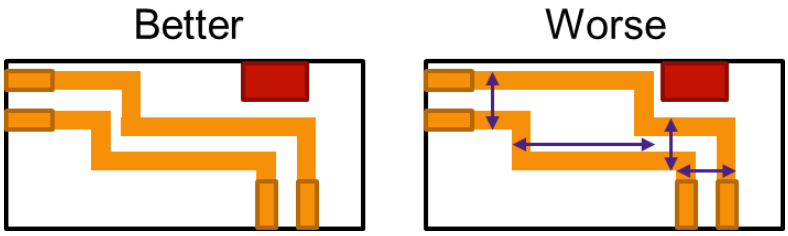

Figure 18 Compactness Cost Example

Overall, in an ideal case, the routing cost of a perfectly routed bus is close to $\alpha + \beta + \gamma$.

C. Penalty Cost

The penalty cost C_P consists of spacing violation penalty P_s and route fail penalty P_f. It is defined as:

$C_P = P_s + P_f$, where

P_s = number of spacing violations $\times$ δ

P_f = number of route fail buses $\times$ ε

For each wire, it should keep at least the distance of the spacing constraint of the layer with the following objects:

- Other wires of different bits.
- Obstacles.
- Design boundary.

Take Figure 19 for example, p_1 should keep at least S_1 with p_2, O_1, and the design boundary. Spacing violations between the two objects will only be counted once. For example, if p_1 did not keep enough spacing with the left edge and the bottom edge of the design boundary, the spacing violation between the two is counted once. Note that the routing paths of the route fail buses still count toward spacing violations if there are any. Contestants should remove floating or failed routing paths. Any violation caused by the floating or failed routing paths counts toward the cost as well. Vias are treated as zero size and will not have spacing violation with other objects. Depending on the values of δ and ε, if a routed bus has too many spacing violations, it may not be worthwhile to leave it routed.

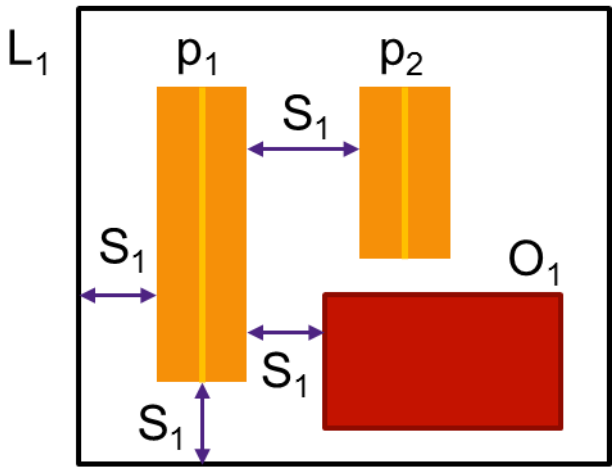

Figure 19 Spacing Violation Example

The rules of determining route success or route fail of a bus is already defined in Section V.A. If a bus route fail, the route fail penalty ε is added to the penalty cost. Generally, ε is large enough (much larger than $\alpha+\beta+\gamma$) such that the penalty of a failed bus is almost always larger than the routing cost of the reasonably routed bus. Note that large test cases do not guarantee to have a clean solution with no bus fail or violations. Also, sometimes a result with few violations might have smaller cost than that of one with no violations. Contestants need to determine how to optimize the routing results based on the given cost function.

D. Runtime Limit

Each test case has a runtime limit specified in the input file as **wall clock time** in **minutes**. Programs that exceed the runtime limit will be killed. No matter the program is killed or terminates normally, the output file will be evaluated. If no output file is found or the output format is wrong, the overall cost for the test case is treated as if all buses route fail.

E. Multithreading

Multithreading is allowed and encouraged. The maximum number of cores that can be used is 4.

VI. Example Test Case and Execution Steps

Take Figure 2 for example, the input is:

```
RUNTIME 1
ALPHA 5
BETA 1
GAMMA 5
DELTA 8
```

```
EPSILON 200
DESIGN_BOUNDARY (0 0) (1000 1000)
LAYERS 2
L1 vertical 20
L2 horizontal 30
ENDLAYERS
TRACKS 34
L1 (100 0) (100 1000) 10
L1 (140 0) (140 1000) 10
L1 (180 0) (180 1000) 10
L1 (220 0) (220 1000) 10
L1 (260 0) (260 1000) 10
L1 (360 0) (360 1000) 5
L1 (400 0) (400 1000) 5
L1 (440 0) (440 1000) 5
L1 (480 0) (480 1000) 5
L1 (520 0) (520 1000) 5
L1 (620 0) (620 1000) 10
L1 (660 0) (660 1000) 10
L1 (700 0) (700 1000) 10
L1 (740 0) (740 1000) 10
L1 (780 0) (780 1000) 10
L1 (880 0) (880 1000) 5
L1 (920 0) (920 1000) 5
L1 (960 0) (960 1000) 5
L2 (0 50) (1000 50) 10
L2 (0 100) (1000 100) 10
L2 (0 200) (1000 200) 10
L2 (0 250) (1000 250) 10
L2 (0 350) (1000 350) 10
L2 (0 400) (1000 400) 10
L2 (0 450) (1000 450) 10
L2 (0 550) (1000 550) 10
L2 (0 600) (1000 600) 10
L2 (0 650) (1000 650) 10
L2 (0 750) (1000 750) 10
L2 (0 800) (1000 800) 10
L2 (450 500) (1000 500) 10
```

```
L2 (450 700) (1000 700) 10
L2 (450 850) (1000 850) 10
L2 (450 900) (1000 900) 10
ENDTRACKS
BUSES 1
BUS B1
3
2
WIDTH 2
10
10
ENDWIDTH
BIT 0
L2 (0 795) (20 805)
L2 (980 595) (1000 605)
ENDBIT
BIT 1
L2 (0 745) (20 755)
L2 (980 695) (1000 705)
ENDBIT
BIT 2
L2 (0 645) (20 655)
L2 (980 745) (1000 755)
ENDBIT
ENDBUS
ENDBUSES
OBSTACLES 17
L1 (175 245) (185 255)
L1 (175 395) (185 405)
L1 (355 145) (365 155)
L1 (355 295) (365 305)
L1 (355 445) (365 455)
L1 (515 245) (515 255)
L1 (525 395) (525 405)
L1 (735 195) (925 505)
L2 (175 245) (185 255)
L2 (175 395) (185 405)
L2 (355 145) (365 155)
```

```
L2 (355 295) (365 305)
L2 (355 445) (365 455)
L2 (515 245) (515 255)
L2 (525 395) (525 405)
L2 (255 595) (450 900)
L2 (735 195) (925 505)
ENDOBSTACLES
```

The output in Figure 2 would be:

```
BUS B1
BIT 0
PATH 9
L2 (20 800) (220 800)
L1 (220 800)
L1 (220 200) (220 800)
L1 (220 200)
L2 (220 200) (620 200)
L1 (620 200)
L1 (620 200) (620 600)
L1 (620 600)
L2 (620 600) (980 600)
ENDPATH
ENDBIT
BIT 1
PATH 9
L2 (20 800) (140 800)
L1 (140 800)
L1 (140 100) (140 800)
L1 (140 100)
L2 (140 100) (660 100)
L1 (660 100)
L1 (660 100) (660 700)
L1 (660 700)
L2 (660 700) (980 700)
ENDPATH
ENDBIT
BIT 2
PATH 9
```

```
L2 (20 700) (120 700)
L1 (120 700)
L1 (120 50) (120 700)
L1 (120 50)
L2 (120 50) (700 50)
L1 (700 50)
L1 (700 50) (700 750)
L1 (700 750)
L2 (700 750) (980 750)
ENDPATH
ENDBIT
ENDBUS
```

The program should be named as “bus_router” and will be executed as:
`./bus_router <input file name> <output file name>`

VII. Test Cases and Evaluator

Two example test cases are released with the problem announcement in example.tar.gz. The first example case, example_1, is the case in Figure 2. The second example case, example_2, is a case with 5 buses with uniform track configuration. Note that example cases will **NOT** be used in grading.

Other test cases and evaluator will be released afterwards.

VIII. Reference

J. Cong, Jie Fang and Kei-Yong Khoo, "DUNE-a multilayer gridless routing system," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 5, pp. 633-647, May 2001.

X. FAQ

Q1.

The spec told us that the tracks may not cover the whole design, so my question is: Can we make use of those regions which are not cover by the tracks or we can only put wires on the tracks to avoid routing failed?

A1.

Regarding routing on- and off-track is explained in Paragraph 2 in Section V.A “... For the second requirement, any wire off-track or has bus width constraint larger than the track width constraint is considered as route fail. ...”

In short, off-track routing is considered as route fail, and the penalty of route fail is explained in Section V.C.

The pins are guaranteed to overlap at least 1 legal track, but the total routable region may or may not have valid track solution for routing all the buses.

It is up to the contestant to perform trade-offs to minimize the overall cost.