Introduction of ISPD19 Contest Problem

The 2019 ISPD contest augments the 2018 ISPD initial detailed routing problem by adding more realistic design rules faced by physical design practitioners in the industry. The **new or updated design rules** considered in this year contest are highlighted in ***blue*** in the corresponding session title.
In this contest, the quality of the detailed routing solution is evaluated by the following aspects.

(1) Connectivity constraints
(2) LEF routing rules
(3) Routing preference metrics

According to these constraints/rules/metrics, we will come out a score for a given detailed routing solution, and the ranking of each participated team for the contest will be based on the scores. The details of these constraints/rules/metrics will be introduced in the following sections.

Connectivity Constraints

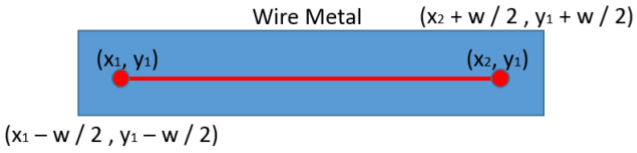
The connectivity constraint has to be satisfied in order to guarantee the valid signal and the routing wires that are able to be implemented. Therefore, the connectivity constraints have the highest priority to be obeyed.

1. OPEN

The pins of each net defined in *.input.def file need to be fully connected. If any pin in a net is disconnected, the net will be considered as an open net and huge score penalty will be applied.

2. SHORT

A wire metal is defined by the center line of a routing wire with a half wire width extension. Figure 1 shows a wire metal formed by a horizontal routing wire from (x1, y1) to (x2, y1) on the layer where wire width is *w*, which is specified by the WIDTH statement on each routing layer in LEF. On the other hand, a via metal is defined by the coordinate of a via with the metal extension defined in *.input.lef file. Figure 1 shows an example of via metal. Note that, ***in this contest, each layer has only one wire width***. Namely, non-default rule is not considered in this contest for simplicity. However, ***each layer may have one or multiple vias for choose to use.*** A short violation will happen when either a via metal or wire metal overlaps with another via metal, wire metal, blockages, or pin shapes. The intersection part between two objects are the short area.




```
VIA VIA45_2ST_S DEFAULT
    LAYER Metal4 ;
        RECT -0.035000 -0.325000 0.035000 0.065000 ;
    LAYER Via4 ;
        RECT -0.035000 -0.175000 0.035000 -0.105000 ;
    LAYER Metal5 ;
        RECT -0.065000 -0.175000 0.065000 0.035000 ;
END VIA45_2ST_S
```

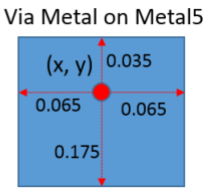


Figure 1

LEF Routing Rules

A detailed router need to consider many routing results defined in LEF files in order to meet the manufacturing requirements from foundries. Different technology nodes, different foundries, and different designs may have different routing rules. Because this contest focuses on the initial detailed routing step, so we only consider the most common and major routing rules. This section will briefly introduce these rules and how their syntax represents in LEF files. If you want to get more details about these routing rules, you can check the following document.
http://ispd.cc/contests/18/lefdefref.pdf

1. SPACINGTABLE

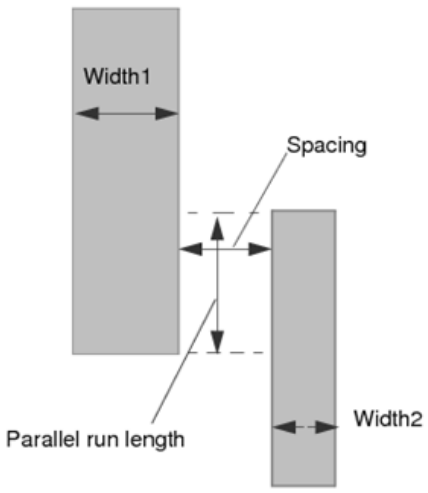
```
LAYER M1
TYPE ROUTING ;
SPACING 0.060000 ;
SPACINGTABLE
   PARALLELRUNLENGTH      0.0000      0.500
   WIDTH          0.0000      0.0600      0.150
   WIDTH          0.1000      0.1000      0.200
   WIDTH          0.7500      0.2500      0.300
   WIDTH          1.5000      0.4500      0.600;
END M1
```

This rule specifies the spacing tables to use for wiring on metal layers. The syntax for describing spacing tables is defined as follows. The *length*, *width*, and *spacing* values must be specified in increasing order.

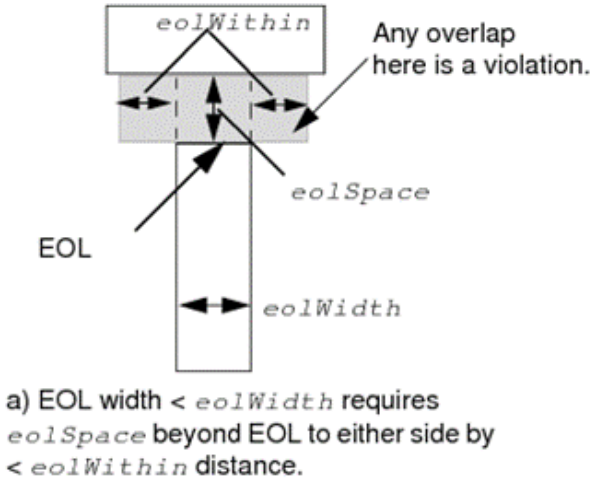
```
SPACINGTABLE
  PARALLELRUNLENGTH {length} ...
    {WIDTH width {spacing} ...}... ;
```

The table specifies the required spacing, in microns. between two objects based on their parallel run length (PRL) and their widths. Figure 2(a) shows an example of two objects running in parallel. In this contest, *length*, *width*, and *spacing* in the table always be non-negative values, and the *spacing* value is always greater than or equal to the default minimum spacing value defined by SPACING syntax. Moreover, if the PRL is greater than the first length value in the table, and the maximum width of the two objects is greater than *width*, then the spacing between the two objects must be greater than or equal to the corresponding *spacing* value. For example, if the maximum width of the two objects is 0.7 and the PRL between them is 0.6, the required spacing will be 0.2 per the above table. Note that, the first-column spacing value is the minimum spacing for a given width, even if the PRL value is not met.

In this year contest, the spacing table will have multiple columns and rows, thus the big spacing may be triggered when two wires run in parallel for long distances. Therefore, long PRL is better to be avoided.



(a)



(b)

Figure 2

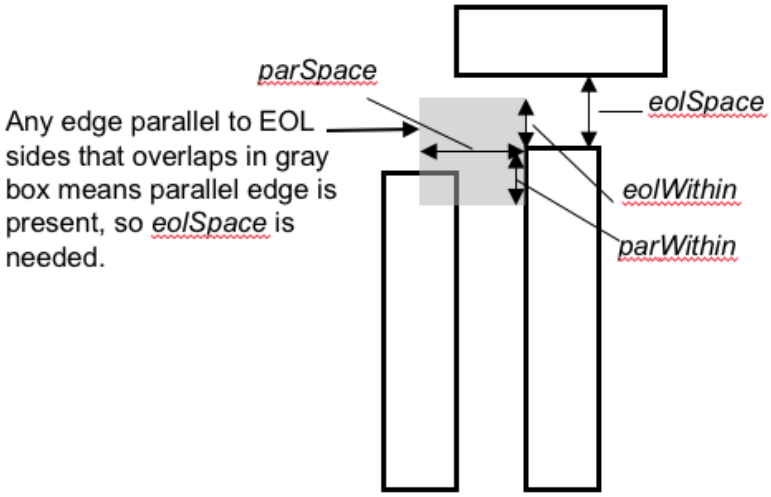


Figure 2 (c)

2. ENDOFLINE

LAYER M1
TYPE ROUTING ;
SPACING 0.090000 ENDOFLINE 0.090000 WITHIN 0.025000
PARALLELEDGE 0.05000 WITHIN 0.025000 ;
END M1

The syntax for describing EOL spacing rules is defined as follows:
SPACING *eolSpace* ENDOFLINE *eolWidth* WITHIN *eolWithin*
[PARALLELEDGE *parSpace* WITHIN *parWithin*];

Indicates that an edge that is shorter than *eolWidth*, noted as end-of-line (EOL from now on) edge requires

spacing greater than or equal to *eolSpace* beyond the EOL anywhere within (that is, less than) *eolWithin* distance (see Figure 2(b)).
Typically, *eolSpace* is slightly larger than the minimum allowed spacing on the layer. The *eolWithin* value must be less than the minimum allowed spacing.
If PARALLELEDGE is defined, the EOL rule is applied only if there is a parallel-edge less than *parSpace* away that is also less than *parWithin* from the end of the wire (see Figure 2(c)).
The PARALLELEDGE search window is constructed by extending from the corner with EOL edge sideward by *parSpace*, extending forward by *eolWithin*, and extending backward by *parWithin*.
If there is no parallel edge present in this search window, then the EOL rule can be ignored for this particular EOL edge.
In this year contest, we will add PARALLELEDGE condition to the EOL to match with the cell definitions used in the design. This PARALLELEDGE condition relaxes the existing EOL rule such that not every EOL edge need to satisfy the *eolSpace* requirement.

3. CUT SPACING

LAYER V1
TYPE CUT ;
SPACING 0.070000 ;
END V1

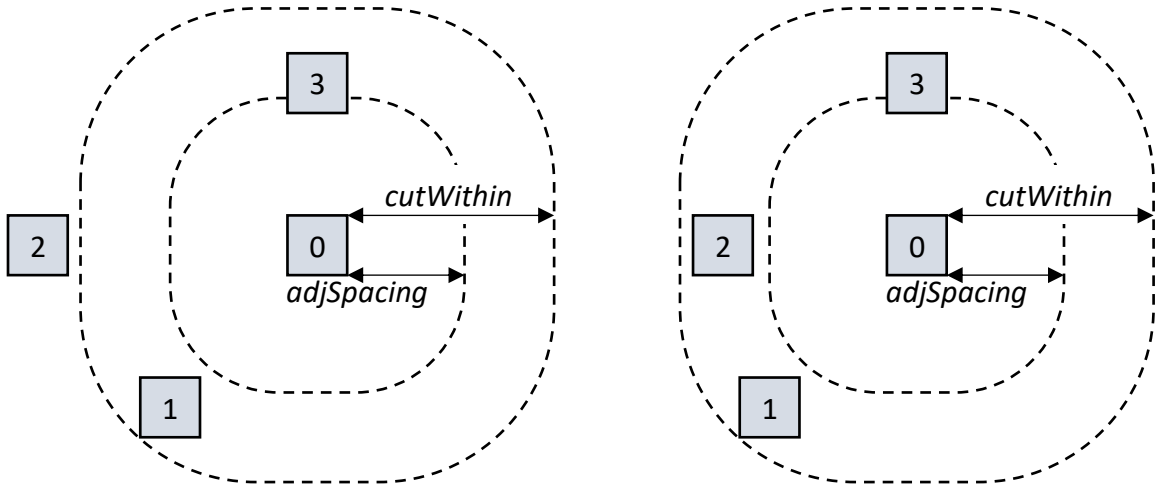
Specifies the minimum spacing allowed between via cuts on the same net or different nets.

4. ADJACENT CUT SPACING

LAYER V1
TYPE CUT ;
SPACING 0.10000 ADJACENTCUTS 3 WITHIN 0.15000 ;
END V1

The syntax for describing ADJACENTCUTS spacing rule is defined as follows:
SPACING *adjSpacing* ADJACENTCUTS {2 | 3 | 4} WITHIN *cutWithin* ;

The adjacent cut rule specifies the minimum spacing allowed between via cuts on the same net or different nets when the cut has two, three, or four via cuts that are less than *cutWithin* distance, in microns, from each other. A cut is considered adjacent if it is within *cutWithin* distance of another cut in any direction (including a 45-degree angle).



(a) **Figure 3** (b)

Figure 3(a) shows a cut (indicated by the number "0") that has three other cuts in its vicinity. Cut #1 and cut #3 are within the *cutWithin* distance but cut #2 is outside this distance. Thus, cut #0 only has two adjacent cuts. If the adjacent cuts rule specifies 3 as the number of required cuts such as follows:

SPACING *adjSpacing* ADJACENTCUTS 3 WITHIN *cutWithin* ;

then this cuts configuration does not trigger any adjacent cut spacing violation because the number of cuts is not satisfied (even if the spacing between cut #0 and cut #3 is less than *adjSpacing*).

On the other hand, Figure 3(b) shows similar configuration as Figure 3(a) but with cut #2 inside the *cutWithin* distance. This brings the number of adjacent cuts to 3 for cut #0. If the same rule is used, then this configuration will yield an adjacent cuts violation because cut #3 has spacing that is less than *adjSpacing*.

5. CORNER-TO-CORNER SPACING

```
LAYER M2
TYPE ROUTING ;
PROPERTY LEF58_CORNERSPACING
        "CORNERSPACING CONVEXCORNER EXCEPTEOL 0.08000
        WIDTH 0.0000 SPACING 0.1200
        WIDTH 0.1000 SPACING 0.2200
        WIDTH 0.2000 SPACING 0.3200 ; " ;
END M2
```

The syntax for describing the corner spacing is defined as follows:

```
PROPERTY LEF58_CORNERSPACING
"CORNERSPACING
  {CONVEXCORNER [EXCEPTEOL eolWidth] }
```

{WIDTH *width* SPACING *spacing* } ...
; " ;

The rule specifies the required spacing in MAXXY style, between a convex corner and any edges. The distance in MAXXY style is measured by the following equations for two rectangular objects R1(lx, ly, ux, uy) and R2(lx, ly, ux, uy).

Let dX = the corner-to-corner distance in X direction = max((R1.lx - R2.ux), (R2.lx - R1.ux))
Let dY = the corner-to-corner distance in Y direction = max((R1.ly - R2.uy), (R2.ly - R1.uy))
Then the MAXXY distance between the two rectangles is: max(dX, dY)

The rule will be triggered when the parallel run length between two objects is less than or equal to 0. If the width of a wire containing the corner is greater than *width* value specified in the table, then the corresponding spacing is applied.

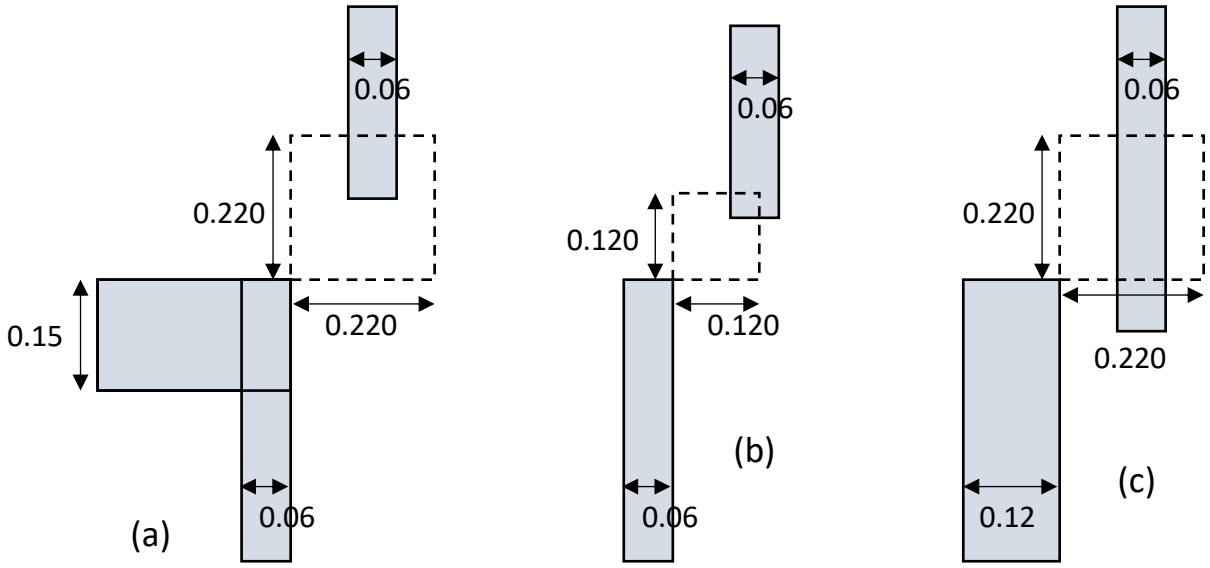


Figure 4

The optional argument of EXCEPTEOL specifies that the corner spacing rule does not apply to a corner connected to a EOL edge whose width is less than the *eolWidth*.

Figure 4 shows three different routing configurations. Using the above corner spacing rule, the routing configuration in Figure 4(a) will have a corner spacing violation. The wide metal (width 0.15) triggers the corner spacing rule thus requires 0.220 spacing measured in MAXXY fashion. Configuration in Figure 4(b) does not have corner spacing violation because the rule is exempted due to end-of-line edge less than 0.080 that is adjacent to the corner (which applies to both the lower metal and the upper metal). Figure 4(c) also shows a routing configuration that does not have any corner spacing violation. In this case, the parallel run length between the convex corner of the wide metal and the nearby wire is greater than 0, thus the corner spacing rule does not apply.

6. MIN AREA

LAYER Metal1
TYPE ROUTING ;

WIDTH 0.06 ;
AREA 0.02 ;

Specifies the minimum metal area required for polygons on the layer. All polygons must have an area that is greater than or equal to *minArea*. *Type:* Float, specified in microns squared

When a routed metal segment is small such that the whole polygon area will not satisfy the min area rule, a patch metal can be added to increase the area for the polygon. See Figure 5(a) to 5(c) for possible patch solution added to the existing metal routing to satisfy min area rule. However, the location and the size of the patch metal must be decided carefully so it will not cause any spacing or short violation. In addition, the overlapping region between patch and the existing metal routing must be greater or equal to the minimum *width* of the current routing layer (see Figure 5(d)).

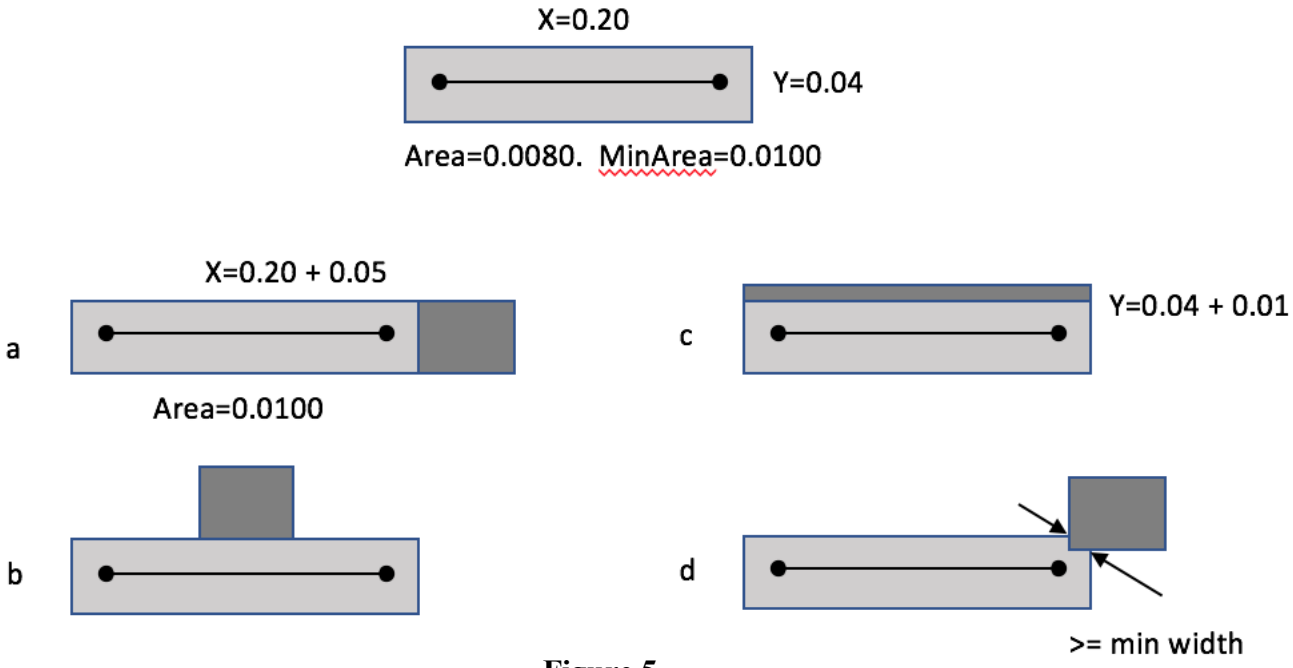


Figure 5

A patch metal is represented by the syntax "RECT (deltax1 deltay1 deltax2 deltay2)" in DEF file, which indicates that a rectangle is created from the previous (x y) routing point using the delta values. The RECT values leave the current point and layer unchanged. For example, in DEF file, a net "net123" has a patch metal (100, 100, 200, 200), which can be represented as "…(120, 110) RECT (-20, -10, 80, 90)".

Routing Preference Metrics

There are several metrics generally used to evaluate a detailed routing solution. Although they are not hard rules, the quality of a routing solution usually could be better in terms of timing, routability, manufacturability if the detailed router considers and optimizes these metrics.

1. Routing Guide Honoring

In the typical routing flow, global routing performs followed by detailed routing. A global routing result is usually well optimized for certain metrics (e.g., congestion, timing, skew, or slew), a detailed router needs to honor the global routing result as much as possible in order to minimize the disturbance to these metrics. In this contest, each benchmark has a *.guide file in which every net associates to a list of rectangles. The list of rectangles is called global routing guide to represent the regions passed by the global routing result of the associated net, and the global routing guide guarantees to cover at least a fully connected detailed routing solution for the net. If the center lines of wires or the coordinate of vias route outside of the guide, they will be considered as guide violations; if wires or vias route inside or just on the boundaries of the rectangles, there is no guide violation. In addition, routing guide honoring does not consider patch metals. Namely, patch metals can put out of guides without the penalty. For example, Figure 6(a) illustrates the guide representation in *.guide file for a net "net123" with two guide rectangles. Figure 6(b) shows a routing solution without guide violations, while Figure 6(c) shows a routing solution with guide violations for both via and wires. The score penalty will be applied based on the number of vias and the length of wires route out of guides.

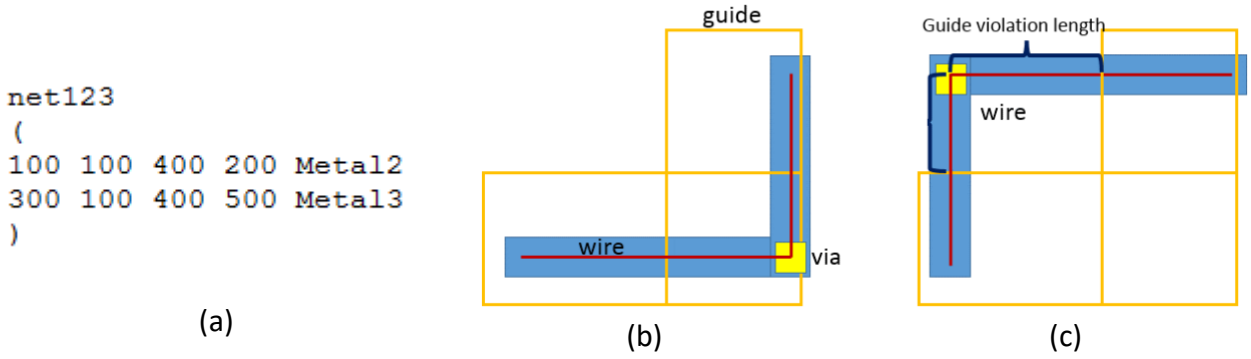


Figure 6

2. Wrong-way Routing

Each metal layer has a preferred routing direction defined by keyword "DIRECTION" in LEF file, which is either HORIZONTAL or VERTICAL. If a wire routes horizontally (vertically) on a vertical (horizontal) layer, the wire is considered as a wrong-way wire. The length of wrong-way wires will contribute the penalty to the scoring function.

3. Off-track Routing

Each metal layer has a track structure defined by keyword "TRACKS" in DEF files. The routing wires that align with tracks is so call on-track wires; otherwise, the wires are off-track wires. Also, a via is considered as an on-track via when the coordinate of the via aligns with the tracks on both its bottom and top layers. The length of off-track wires and the number of off-track vias will be considered as a penalty in the scoring function.

4. Double-cut via insertion

For reality and performance concerns, detailed routers prefer to use double-cut/multi-cut vias rather than single-cut vias. This contest will provide both double-cut and single-cut vias in the via library. During solution evaluation, single-cut vias will be more expansive than double-cut vias in order to encourage the usage of double-cut vias. In addition, for some situations, min-area rule can be satisfied by using double-cut vias carefully.

5. Multithreading Determinism

When technology nodes advance and design scale increases, the multithreading framework becomes an important feature to a detailed router. In this contest, we will evaluate the detailed routers on a machine with at least 8 CPUs, so multithreading implementation is encouraged but optional. (It is totally fine if the proposed detailed router uses only a single thread) However, multithreading technique is easier to have non-deterministic. Because non-deterministic behavior is a headache to debug and maintain a detailed router, it is better to avoid that. ***During solution evaluation stage, we will run the proposed detailed router multiple times with the fixed number of threads. If different runs for the same benchmark generate different results, the median result will be considered for the scoring and certain-level of the penalty will be applied to the score.*** The details of the machine status will be announced on 12/31/2018.

Summary

This contest will consider the following constraints/rules/metrics.

1. Number of open nets
2. Short metal area
3. Number of spacing violations (including spacing table, EOL, and cut spacing violations)
4. Number of min-area violations
5. Determinism
6. Total length of the wires outside of the routing guides
7. Total number of the vias outside of routing guides
8. Total length of off-track wires
9. Total number of off-track vias
10. Total length of wrong-way wires
11. Total length of wires
12. Total number of single-cut vias
13. Total number of double-cut vias
14. Total area of patch metals

In this contest, we will use Innovus to verify the detailed routing solutions and report the violations of connectivity and routing rules. The contest provides the document to introduce how to install Innovus and how to use Innovus to evaluate detailed routing solutions. Thus, the contestants can evaluate their detailed routing solution on their local servers. Then, we will provide an evaluation tool to read the violation reports from Innovus and consider the routing preference metrics for the given detailed routing solution to come out a score. The scoring function and the evaluation tool will be released by 12/31/2018.

The runtime of detailed routers will also be considered. There is a runtime factor that considers CPU time and real time to weight the score obtained by the evaluation tool. The contest ranking will be based on the weighted score. In addition, ***this contest will have max runtime and max memory usage constraints. If the usage of either runtime or memory is over a certain threshold for running a benchmark, the benchmark will be considered as a failure.*** More details will be announced by 12/31/2018.