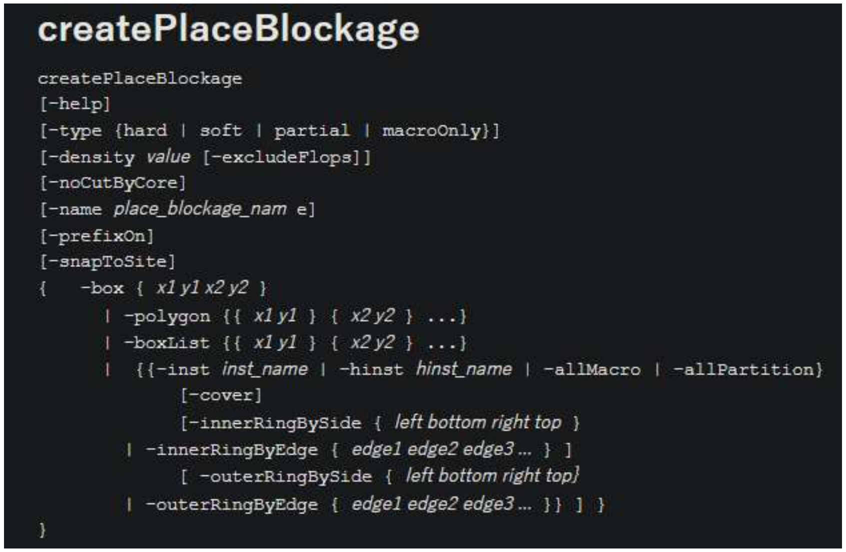
EE201A Project

Justin La

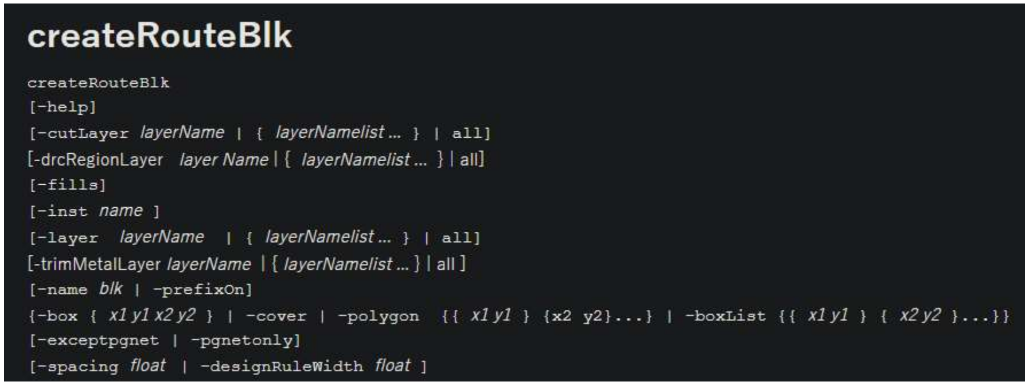
Place and Route Blockage Commands

- Innovus provides commands createRouteBlk and createPlaceBlockage
- createRouteBlk: prevents routing of specified metal layers, signal routes, and hierarchical instances
- createPlaceBlockage: cell placement blockages



createPlaceBlockage

```
createPlaceBlockage
[-help]
[-type {hard | soft | partial | macroOnly}]
[-density value [-excludeFlops]]
[-noCutByCore]
[-name place_blockage_nam e]
[-prefixOn]
[-snapToSite]
{   -box { x1 y1 x2 y2 }
      | -polygon {{ x1 y1 } { x2 y2 } ...}
      | -boxList {{ x1 y1 } { x2 y2 } ...}
      |  {{-inst inst_name | -hinst hinst_name | -allMacro | -allPartition}
             [-cover]
             [-innerRingBySide { left bottom right top }
        | -innerRingByEdge { edge1 edge2 edge3 ... } ]
             [ -outerRingBySide { left bottom right top}
        | -outerRingByEdge { edge1 edge2 edge3 ... }} ] }
}
```



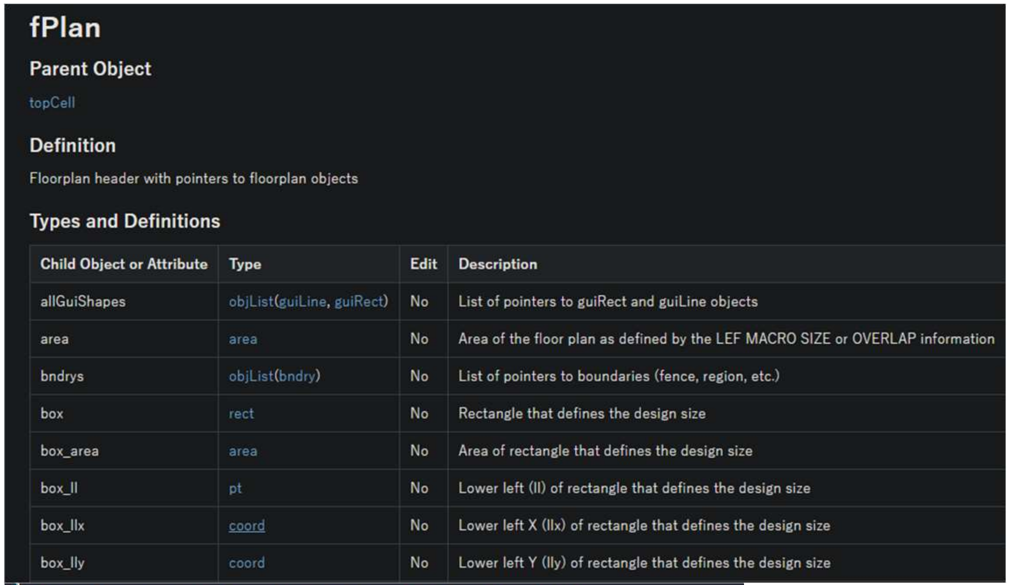
createRouteBlk

```
createRouteBlk
[-help]
[-cutLayer layerName | { layerNamelist ... } | all]
[-drcRegionLayer  layer Name | { layerNamelist ... } | all]
[-fills]
[-inst name ]
[-layer  layerName  | { layerNamelist ... } | all]
[-trimMetalLayer layerName | { layerNamelist ... } | all ]
[-name blk | -prefixOn]
{-box { x1 y1 x2 y2 } | -cover | -polygon  {{ x1 y1 } {x2 y2}...} | -boxList {{ x1 y1 } { x2 y2 }...}}
[-exceptpgnet | -pgnetonly]
[-spacing float | -designRuleWidth float ]
```

- Both blockages allows for defining a bound box to place the blockage. It takes in 2 coordinates, lower left (x,y) and upper right (x,y).
- Place Blockage is selected to use –type hard as to prevent placement of blocks/cells completely. Other options only prevent placement during specific stages
- Route Blockage allows for the input of layers to place blockages.

Box Area

- Finding the box area is required to allow placement of blockages based on percentage of area
  - 40% - 60% range requirement
- “dbGet top.fPlan.box_???” can be used to find the bounding area of the design for both the design and core area.
  - returns lower left and upper right coordinates
- Design area is used for route blockages
- Core area is used for placement blockages.



fPlan

Parent Object

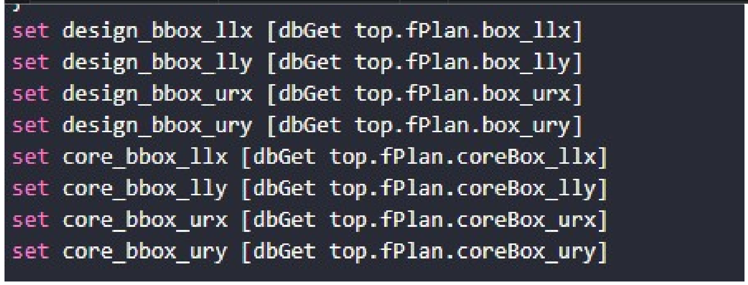
topCell

Definition

Floorplan header with pointers to floorplan objects

Types and Definitions

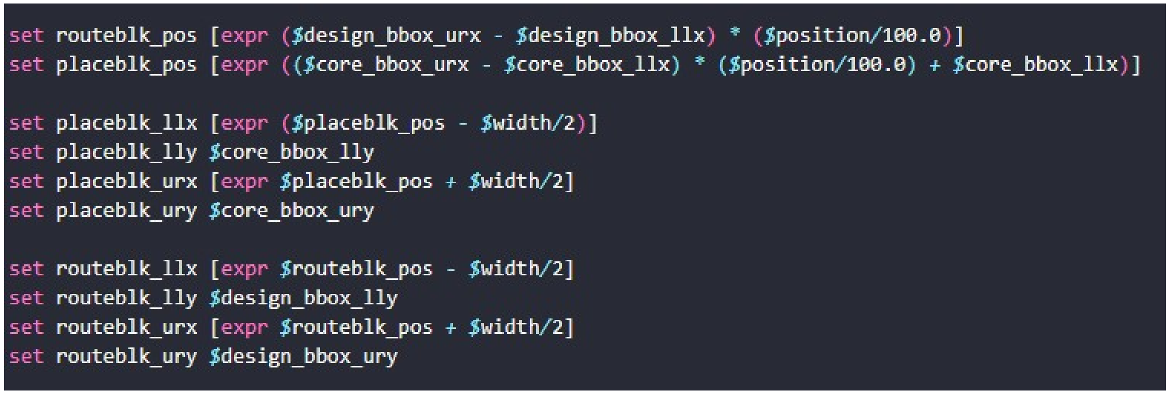
Child Object or Attribute	Type	Edit	Description
allGuiShapes	objList(guiLine, guiRect)	No	List of pointers to guiRect and guiLine objects
area	area	No	Area of the floor plan as defined by the LEF MACRO SIZE or OVERLAP information
bndrys	objList(bndry)	No	List of pointers to boundaries (fence, region, etc.)
box	rect	No	Rectangle that defines the design size
box_area	area	No	Area of rectangle that defines the design size
box_ll	pt	No	Lower left (ll) of rectangle that defines the design size
box_llx	coord	No	Lower left X (llx) of rectangle that defines the design size
box_lly	coord	No	Lower left Y (lly) of rectangle that defines the design size



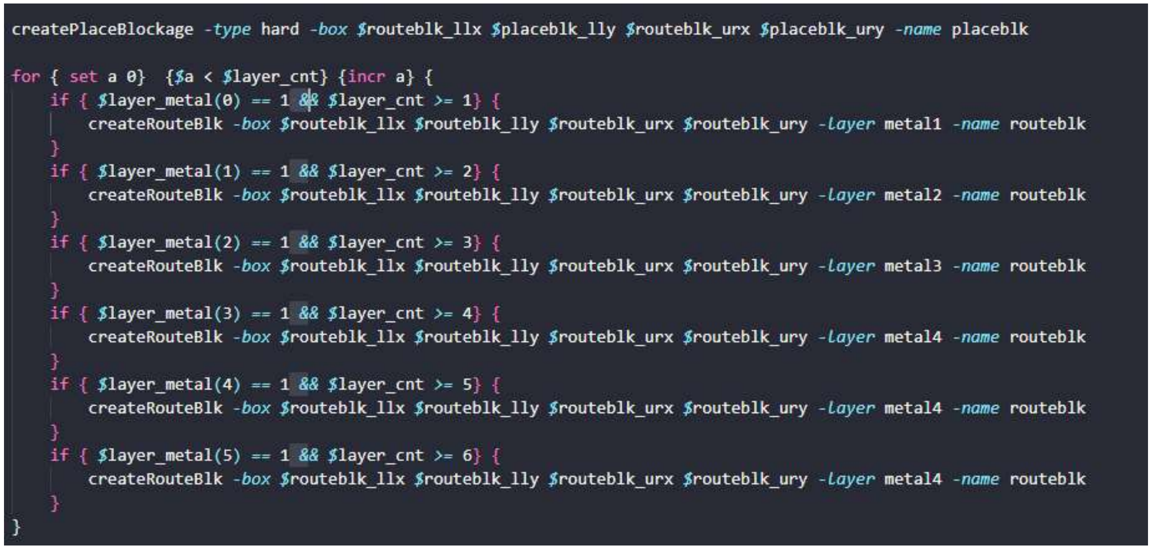
```
set design_bbox_llx [dbGet top.fPlan.box_llx]
set design_bbox_lly [dbGet top.fPlan.box_lly]
set design_bbox_urx [dbGet top.fPlan.box_urx]
set design_bbox_ury [dbGet top.fPlan.box_ury]
set core_bbox_llx [dbGet top.fPlan.coreBox_llx]
set core_bbox_lly [dbGet top.fPlan.coreBox_lly]
set core_bbox_urx [dbGet top.fPlan.coreBox_urx]
set core_bbox_ury [dbGet top.fPlan.coreBox_ury]
```

Blockage Implementation

- The units of the coordinates have been determined to be 1 micron
  - $width = 1
- Position of blockage is placed in the middle with .5 micron and each side for coordinates



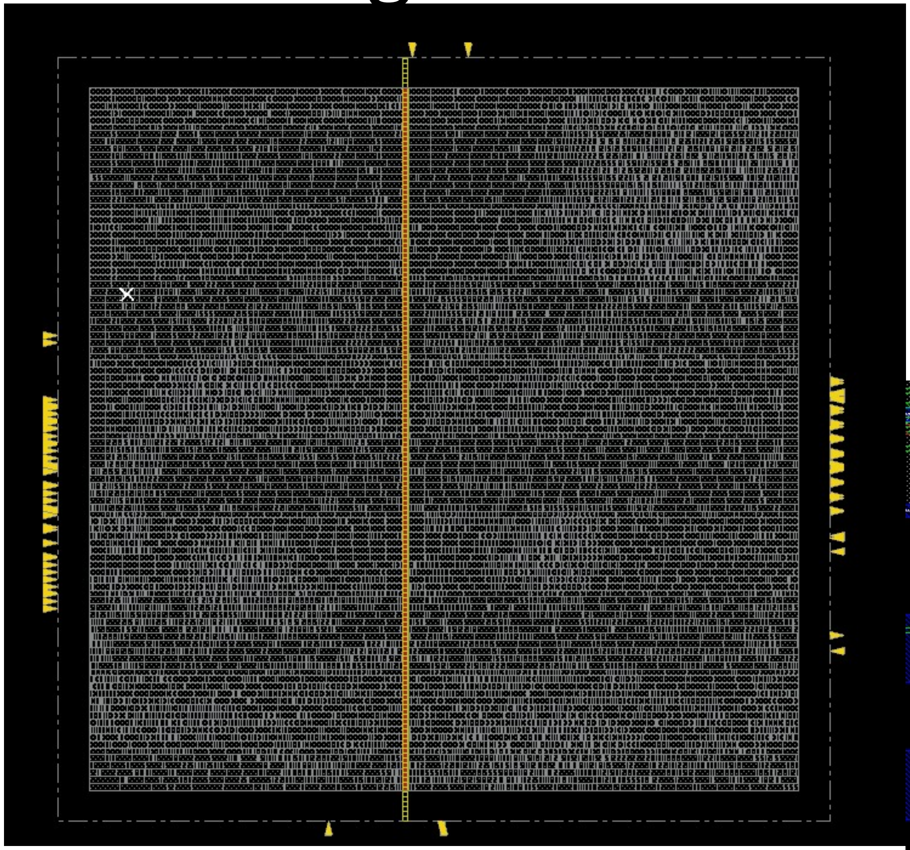
```
set routeblk_pos [expr ($design_bbox_urx - $design_bbox_llx) * ($position/100.0)]
set placeblk_pos [expr (($core_bbox_urx - $core_bbox_llx) * ($position/100.0) + $core_bbox_llx)]

set placeblk_llx [expr ($placeblk_pos - $width/2)]
set placeblk_lly $core_bbox_lly
set placeblk_urx [expr $placeblk_pos + $width/2]
set placeblk_ury $core_bbox_ury

set routeblk_llx [expr $routeblk_pos - $width/2]
set routeblk_lly $design_bbox_lly
set routeblk_urx [expr $routeblk_pos + $width/2]
set routeblk_ury $design_bbox_ury
```



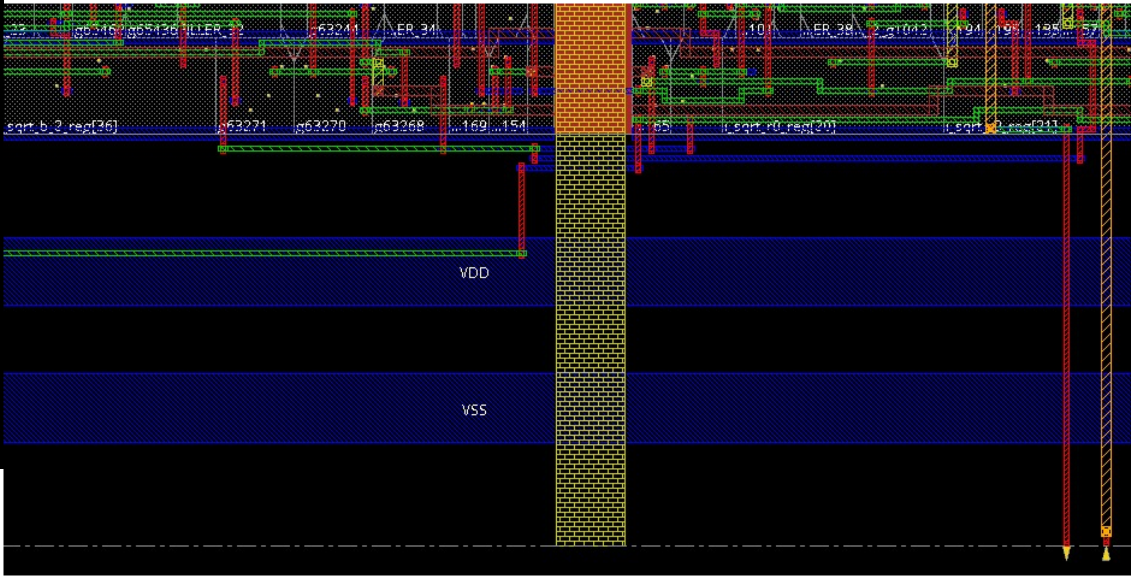
```
createPlaceBlockage -type hard -box $routeblk_llx $placeblk_lly $routeblk_urx $placeblk_ury -name placeblk

for { set a 0}  {$a < $layer_cnt} {incr a} {
    if { $layer_metal(0) == 1 && $layer_cnt >= 1} {
        createRouteBlk -box $routeblk_llx $routeblk_lly $routeblk_urx $routeblk_ury -layer metal1 -name routeblk
    }
    if { $layer_metal(1) == 1 && $layer_cnt >= 2} {
        createRouteBlk -box $routeblk_llx $routeblk_lly $routeblk_urx $routeblk_ury -layer metal2 -name routeblk
    }
    if { $layer_metal(2) == 1 && $layer_cnt >= 3} {
        createRouteBlk -box $routeblk_llx $routeblk_lly $routeblk_urx $routeblk_ury -layer metal3 -name routeblk
    }
    if { $layer_metal(3) == 1 && $layer_cnt >= 4} {
        createRouteBlk -box $routeblk_llx $routeblk_lly $routeblk_urx $routeblk_ury -layer metal4 -name routeblk
    }
    if { $layer_metal(4) == 1 && $layer_cnt >= 5} {
        createRouteBlk -box $routeblk_llx $routeblk_lly $routeblk_urx $routeblk_ury -layer metal4 -name routeblk
    }
    if { $layer_metal(5) == 1 && $layer_cnt >= 6} {
        createRouteBlk -box $routeblk_llx $routeblk_lly $routeblk_urx $routeblk_ury -layer metal4 -name routeblk
    }
}
```

- Both place and route use same x coordinates, so they stack on top of each other
- Only the y coordinates are unique for different area covergae (design vs core area)
- By reading a file, position and layers blocked are fed into the tcl and executed.

Blockages Results

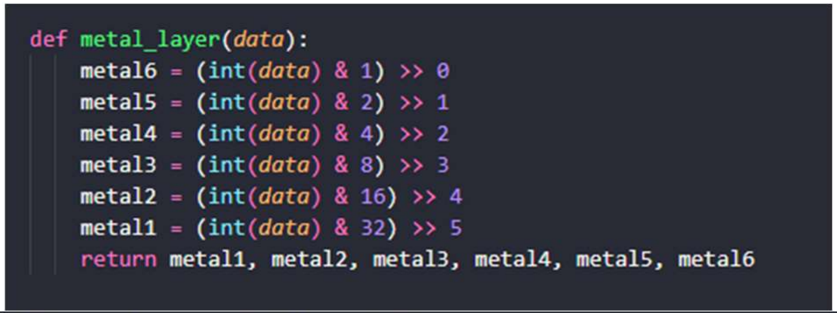


- These are the results of the blockages at 45%
- Bottom picture shows that the route blockage (yellow) extends to the bottom of the design, while place blockage (red) ends at the core area.
- In code, metal 1 is always active. Since horizontal power rings are placed on metal1 they show not to be blocked.

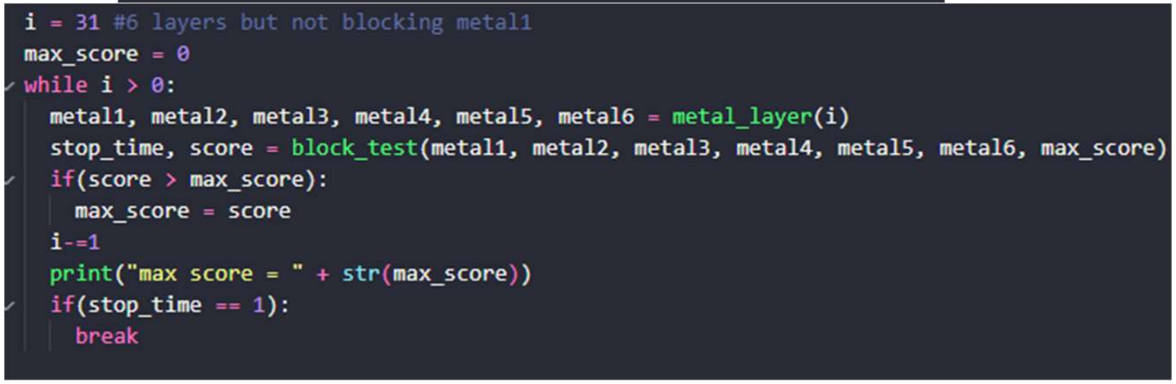


Strategy

- Every variation of metal layer blockage combination can be determined by using binary counting
- Metal1 is the MSB, by only inputting 31 and lower, metal1 will never be blocked.
- By counting down, the idea is to begin testing by running with most aggressive blocked layers first (5 layers blocked)
    - Due to time constraint



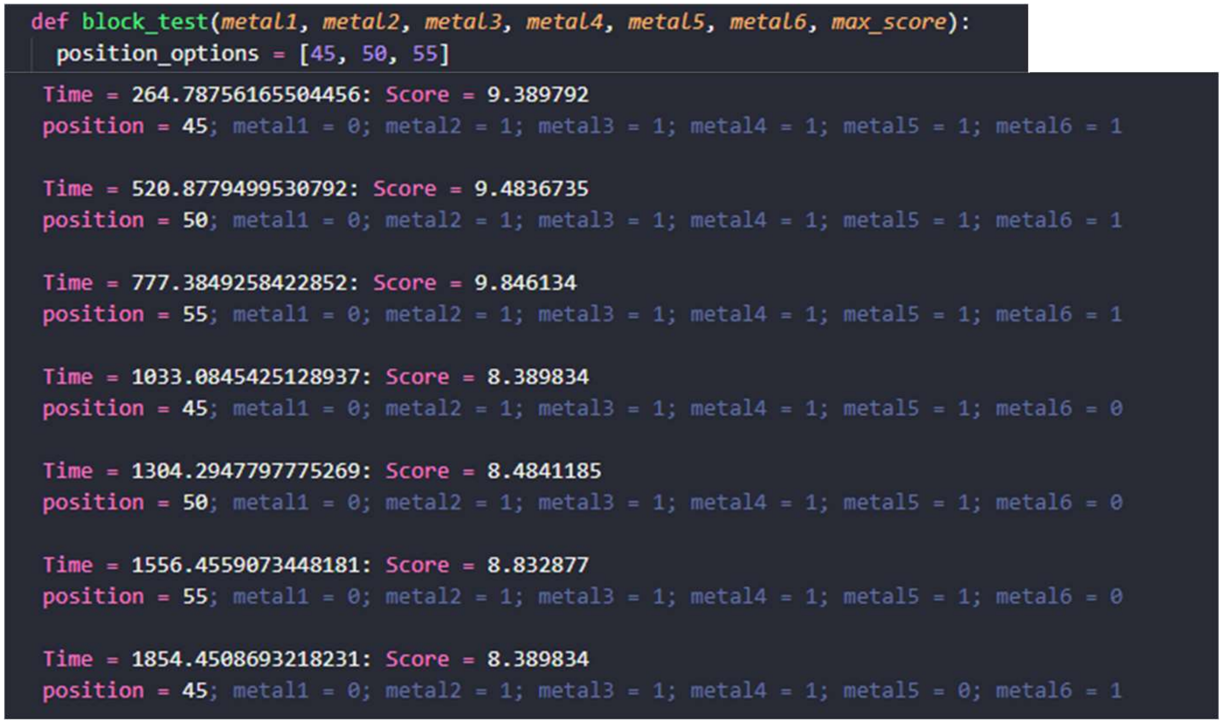
```
def metal_layer(data):
    metal6 = (int(data) & 1) >> 0
    metal5 = (int(data) & 2) >> 1
    metal4 = (int(data) & 4) >> 2
    metal3 = (int(data) & 8) >> 3
    metal2 = (int(data) & 16) >> 4
    metal1 = (int(data) & 32) >> 5
    return metal1, metal2, metal3, metal4, metal5, metal6
```



```
i = 31 #6 layers but not blocking metal1
max_score = 0
while i > 0:
  metal1, metal2, metal3, metal4, metal5, metal6 = metal_layer(i)
  stop_time, score = block_test(metal1, metal2, metal3, metal4, metal5, metal6, max_score)
  if(score > max_score):
    max_score = score
  i-=1
  print("max score = " + str(max_score))
  if(stop_time == 1):
    break
```

Strategy(2)

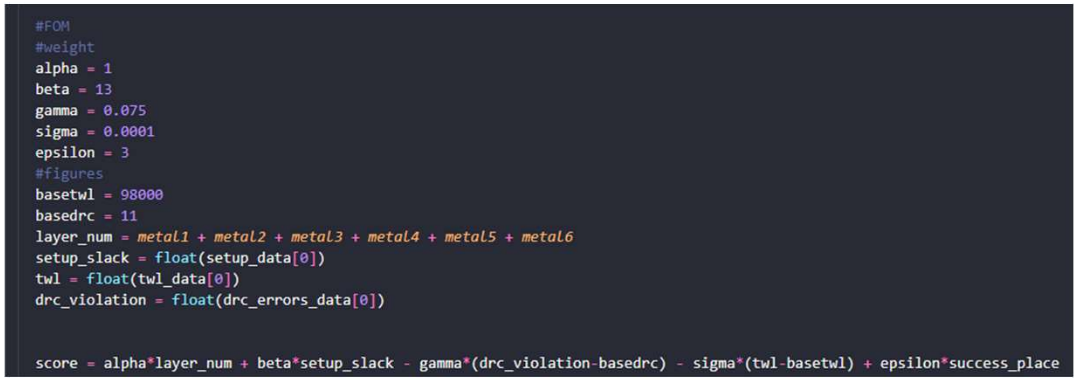
- Average time between runs is about 4 – 5 mins for aggressive blockages.
- With a 1.5hr limit, only about 18-22 runs can be ran.
- Since metal layer blockages place more value in FOM than position, testing for more metal layer combinations was valued more than position of blockage.
- Therefore only 3 positions were chosen: 45, 50, 55.
  - Theses values are closer to the center and will yield less DRC errors, in theory.



```
def block_test(metal1, metal2, metal3, metal4, metal5, metal6, max_score):
  position_options = [45, 50, 55]
```

```
Time = 264.78756165504456: Score = 9.389792
position = 45; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 1

Time = 520.8779499530792: Score = 9.4836735
position = 50; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 1

Time = 777.3849258422852: Score = 9.846134
position = 55; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 1

Time = 1033.0845425128937: Score = 8.389834
position = 45; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 0

Time = 1304.2947797775269: Score = 8.4841185
position = 50; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 0

Time = 1556.4559073448181: Score = 8.832877
position = 55; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 0

Time = 1854.4508693218231: Score = 8.389834
position = 45; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 0; metal6 = 1
```

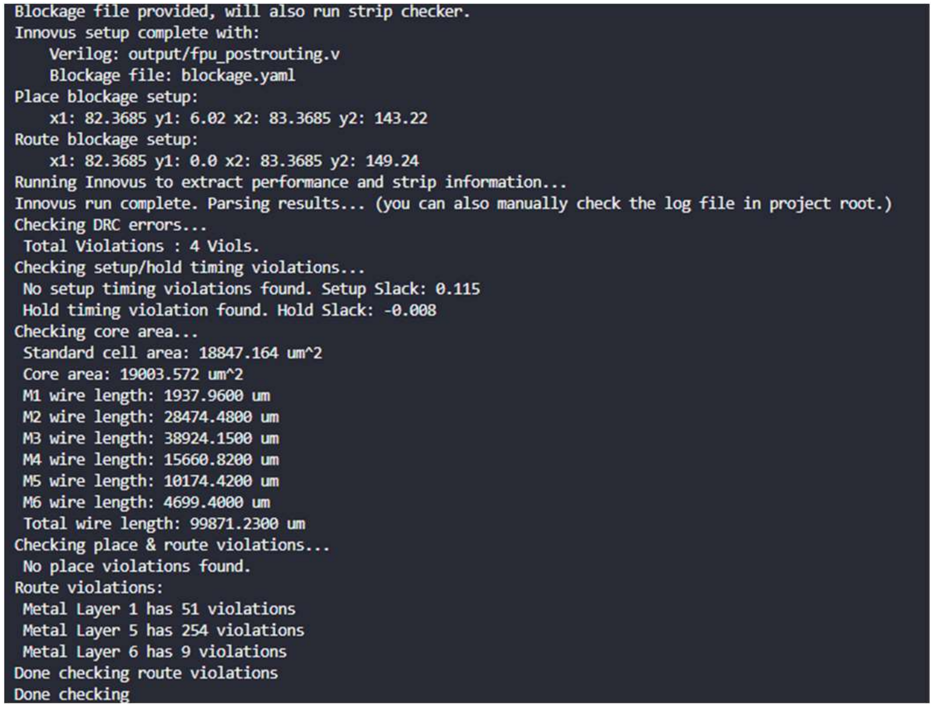
FOM – Figure of Merit



```
#FOM
#weight
alpha = 1
beta = 13
gamma = 0.075
sigma = 0.0001
epsilon = 3
#figures
basetwl = 98000
basedrc = 11
layer_num = metal1 + metal2 + metal3 + metal4 + metal5 + metal6
setup_slack = float(setup_data[0])
twl = float(twl_data[0])
drc_violation = float(drc_errors_data[0])


score = alpha*layer_num + beta*setup_slack - gamma*(drc_violation-basedrc) - sigma*(twl-basetwl) + epsilon*success_place
```

- Due to the way that FOM was determined, using the provided checker as part of the testing process was required.
  - “Successful placement” is only found in the checker
- This requires 2 runs of Innovus to complete for 1 data set

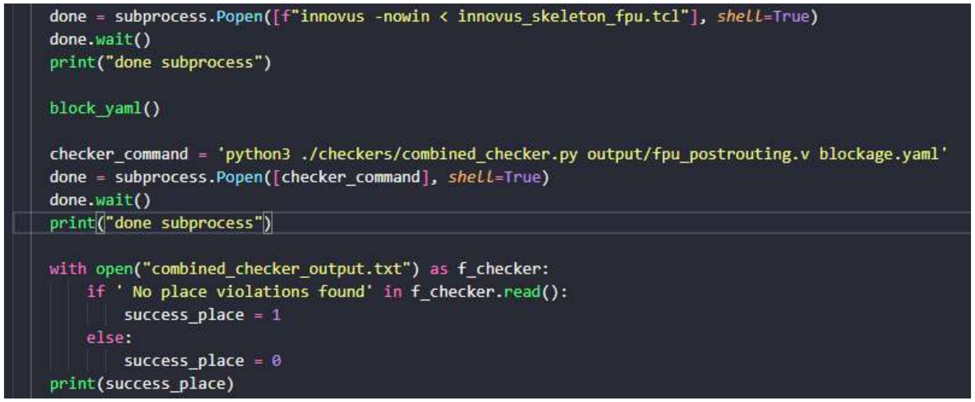
- Checker provides data used for FOM
- setup slack, drc violation, total wirelength, success placement
- Layer number was determined in the code



```
Blockage file provided, will also run strip checker.
Innovus setup complete with:
    Verilog: output/fpu_postrouting.v
    Blockage file: blockage.yaml
Place blockage setup:
    x1: 82.3685 y1: 6.02 x2: 83.3685 y2: 143.22
Route blockage setup:
    x1: 82.3685 y1: 0.0 x2: 83.3685 y2: 149.24
Running Innovus to extract performance and strip information...
Innovus run complete. Parsing results... (you can also manually check the log file in project root.)
Checking DRC errors...
 Total Violations : 4 Viols.
Checking setup/hold timing violations...
 No setup timing violations found. Setup Slack: 0.115
 Hold timing violation found. Hold Slack: -0.008
Checking core area...
 Standard cell area: 18847.164 um^2
 Core area: 19003.572 um^2
 M1 wire length: 1937.9600 um
 M2 wire length: 28474.4800 um
 M3 wire length: 38924.1500 um
 M4 wire length: 15660.8200 um
 M5 wire length: 10174.4200 um
 M6 wire length: 4699.4000 um
 Total wire length: 99871.2300 um
Checking place & route violations...
 No place violations found.
Route violations:
 Metal Layer 1 has 51 violations
 Metal Layer 5 has 254 violations
 Metal Layer 6 has 9 violations
Done checking route violations
Done checking
```

Operation

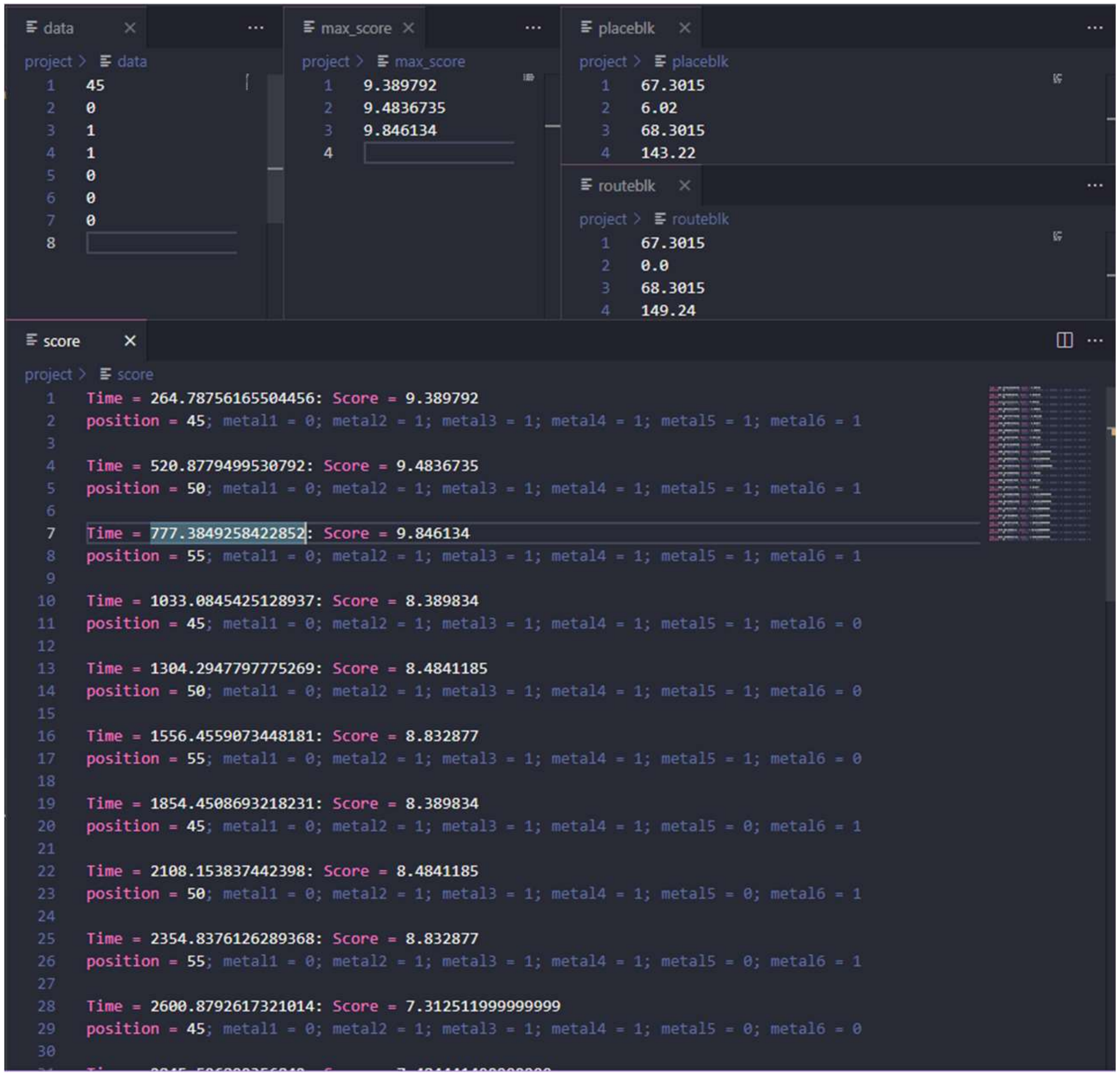
1. block_test receives metal layer to block and the current max score
2. block_test writes position and layer blockage to file “data”
3. subprocess runs Innovus
4. In Innovus, the position and layer blockage information is extracted from ‘data’ and executed
5. After Innovus completes, blockage information is sent to yaml for checker
6. subprocess runs provided checker script
7. checker log is parsed for FOM data
8. subprocess cleans, file with clean.sh and clean_checker.sh
9. block_test loops for different metal layer blockage combination until time limit is reached (5400 sec = 1.5hrs)



```
done = subprocess.Popen([f"innovus -nowin < innovus_skeleton_fpu.tcl"], shell=True)
done.wait()
print("done subprocess")

block_yaml()

checker_command = 'python3 ./checkers/combined_checker.py output/fpu_postrouting.v blockage.yaml'
done = subprocess.Popen([checker_command], shell=True)
done.wait()
print("done subprocess")

with open("combined_checker_output.txt") as f_checker:
    if ' No place violations found' in f_checker.read():
        success_place = 1
    else:
        success_place = 0
print(success_place)
```

Generated Files

- The following files are generated by the python code.
  - data: position and metal layer blockage
  - placeblk: coordinates of placement blockage
  - routeblk: coordinates of route blockage
  - score: FOM score with time elapsed. Also includes information from ‘data’ file
  - max_score: provides the maximum score achieved



```
45
0
1
1
0
0
0
```

```
9.389792
9.4836735
9.846134
```

```
67.3015
6.02
68.3015
143.22
```

```
67.3015
0.0
68.3015
149.24
```

```
Time = 264.78756165504456: Score = 9.389792
position = 45; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 1

Time = 520.8779499530792: Score = 9.4836735
position = 50; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 1

Time = 777.3849258422852: Score = 9.846134
position = 55; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 1

Time = 1033.0845425128937: Score = 8.389834
position = 45; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 0

Time = 1304.2947797775269: Score = 8.4841185
position = 50; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 0

Time = 1556.4559073448181: Score = 8.832877
position = 55; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 1; metal6 = 0

Time = 1854.4508693218231: Score = 8.389834
position = 45; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 0; metal6 = 1

Time = 2108.153837442398: Score = 8.4841185
position = 50; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 0; metal6 = 1

Time = 2354.8376126289368: Score = 8.832877
position = 55; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 0; metal6 = 1

Time = 2600.8792617321014: Score = 7.312511999999999
position = 45; metal1 = 0; metal2 = 1; metal3 = 1; metal4 = 1; metal5 = 0; metal6 = 0
```