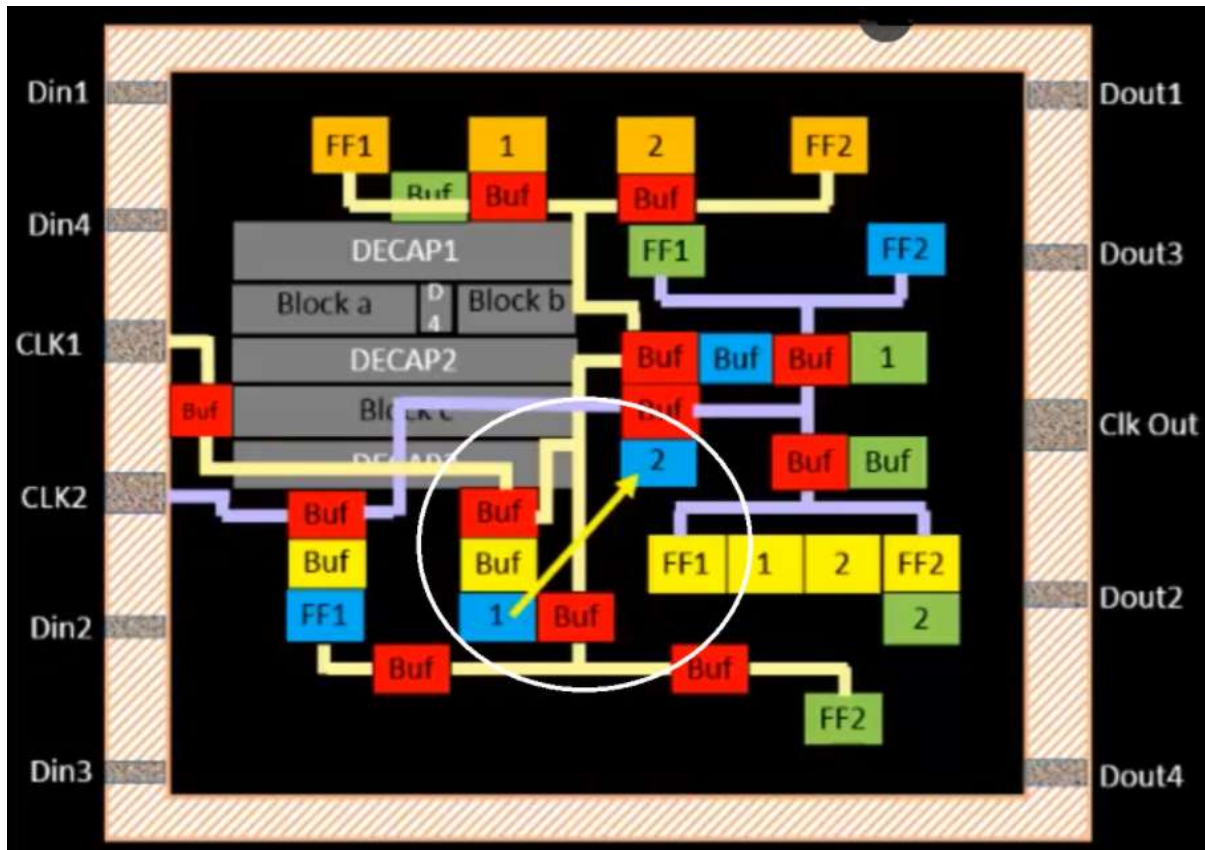


SKY130 DAY 5: Final Steps For RTL2GDS Using TritonRoute and OpenSTA

Routing and Design Check [DRC]

Introduction to Maze Routing and Lee's Algorithm

Routing finds the best connection between elements (e.g., clocks, flip-flops). Algorithms like Steiner Tree and Line Search exist, and one such algorithm is Maze Routing - Lee's Algorithm (Lee, 1961).

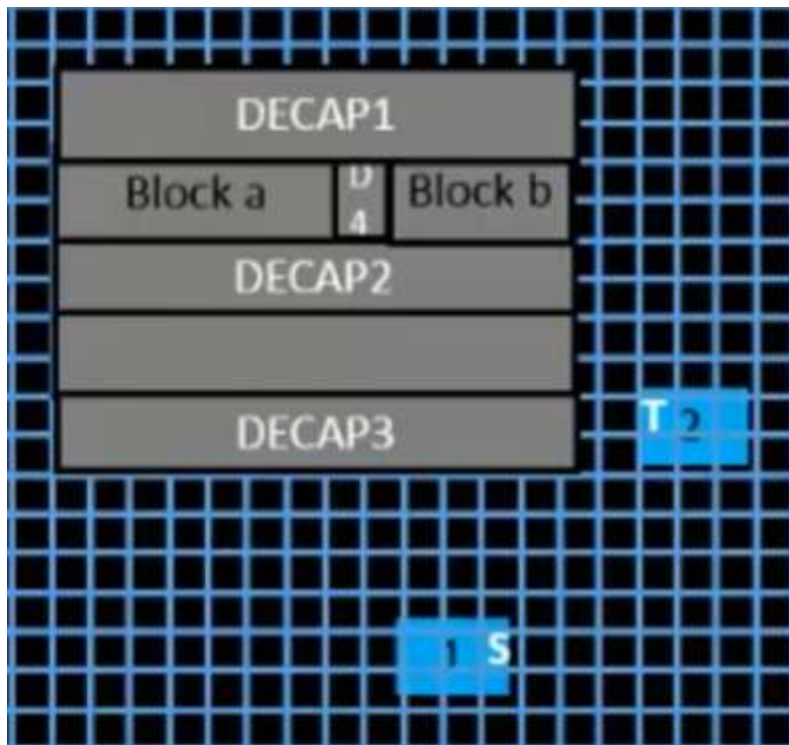


To connect points 1 (source) and 2 (target), the goal is to find the shortest path with minimal zig-zags, usually L-shaped. The algorithm searches for and connects the points, while physically, it represents a wire allowing signals to travel.

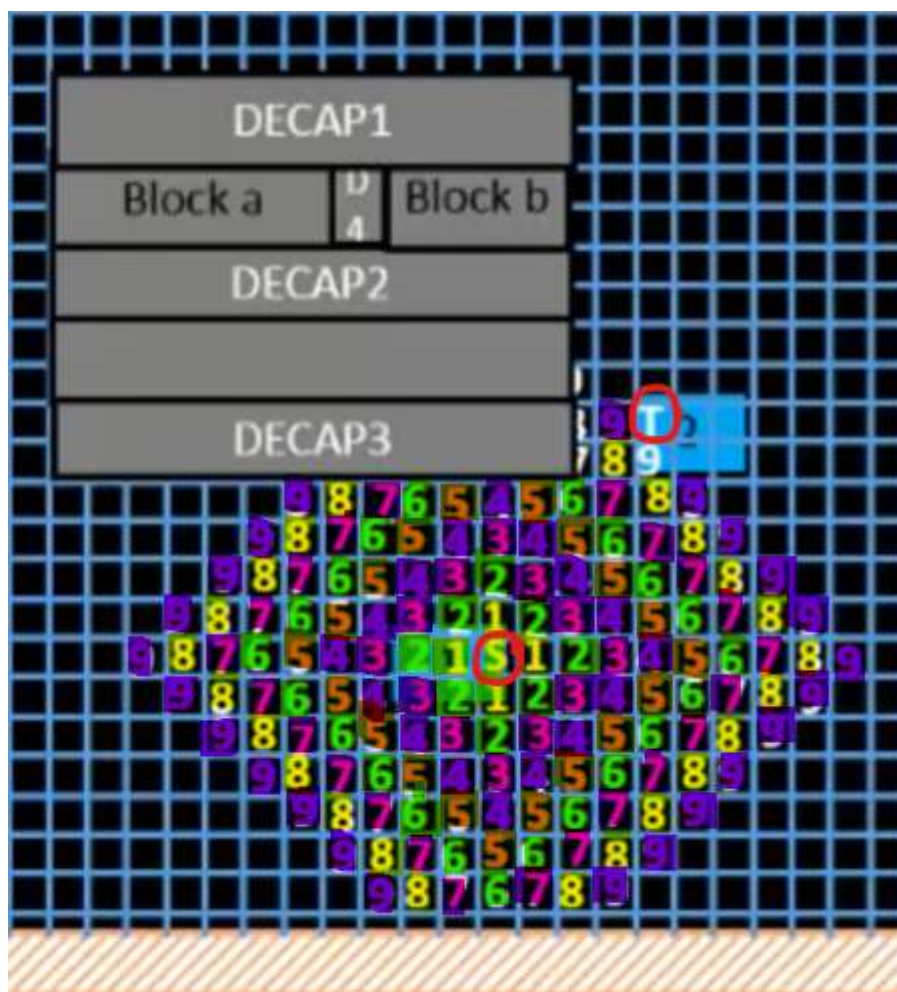
Lee's Algorithm is used in maze routing, where a path needs to be found from source to destination in a grid. It's especially effective for grid or mesh-based structures, making it ideal for integrated circuit design.

Steps:

1. Initialisation - In the initialization step, a routing grid is created and each cell is categorized into one of these states: (i) obstacle, (ii) empty, (iii) visited, (iv) source, (v) target.



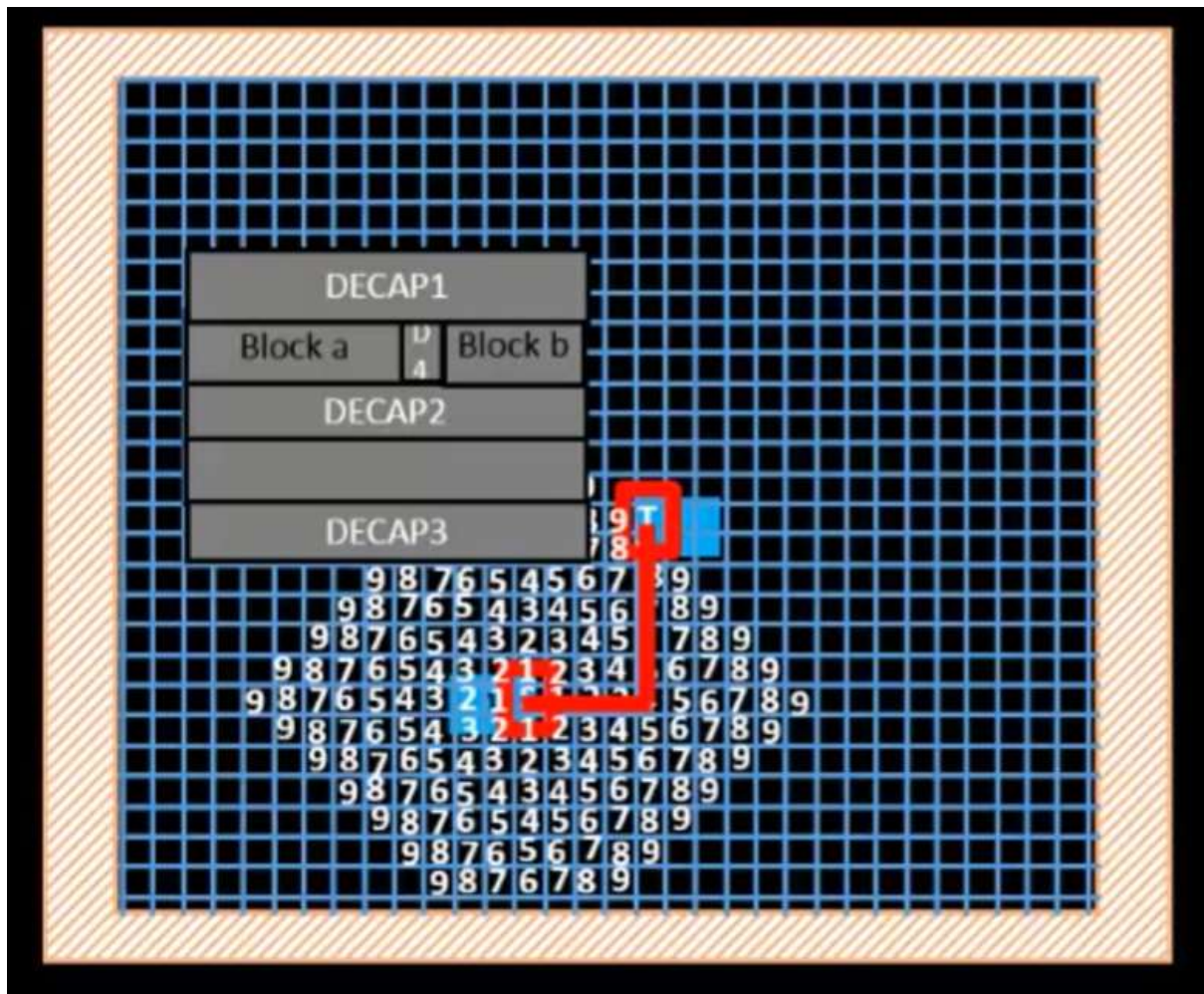
2. Wave Expansion - In the wave expansion step, the algorithm spreads out from the source ('S') in all directions. It checks neighboring cells (up, down, left, right) and assigns a value one greater than the minimum value of its neighbors (excluding obstacles), starting from 1. This continues until the target ('T') is reached or no more cells can be visited.



(Lee's Algorithm Conclusion)

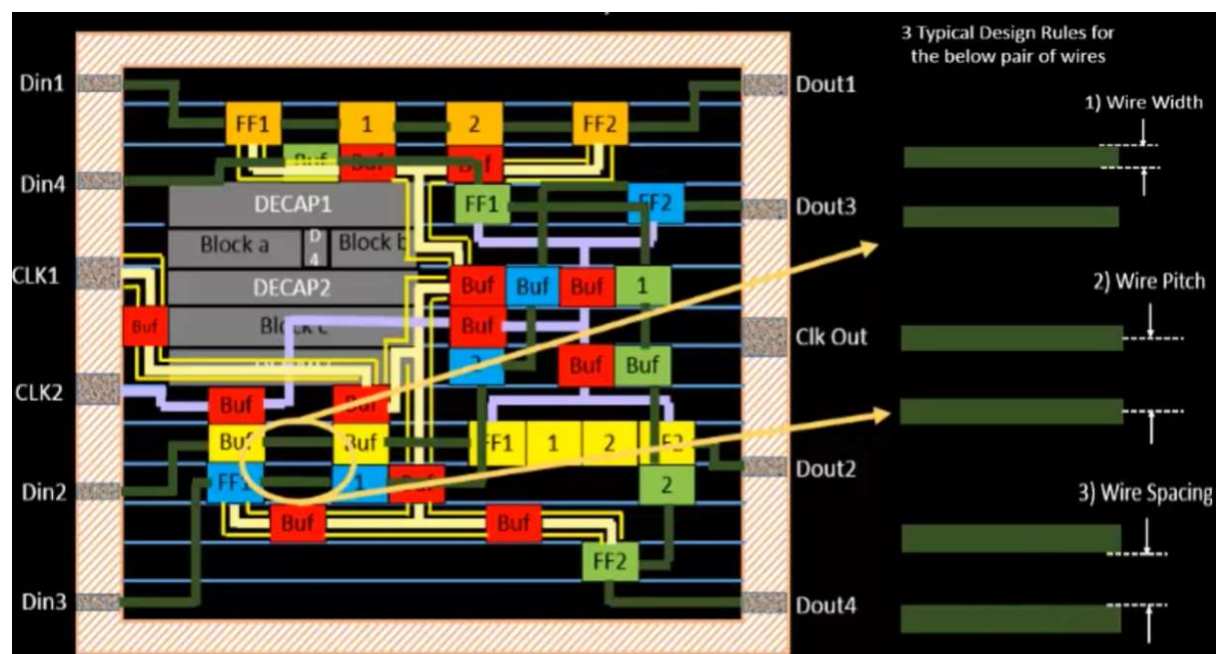
- After reaching the target, the algorithm backtracks to the source by following the cell values. Multiple paths may be found, but the tool selects the one with fewer bends, ensuring the shortest route.

The route should not be diagonal and must not overlap any blockage/obstruction such as macros or HIPs.

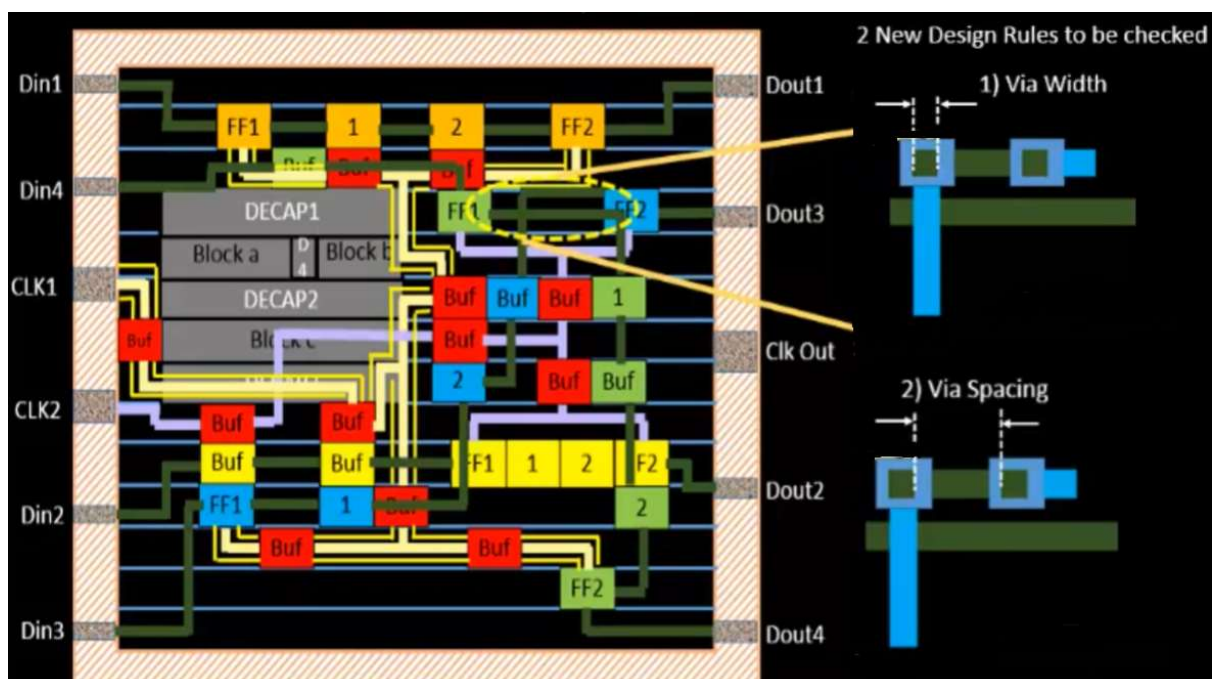
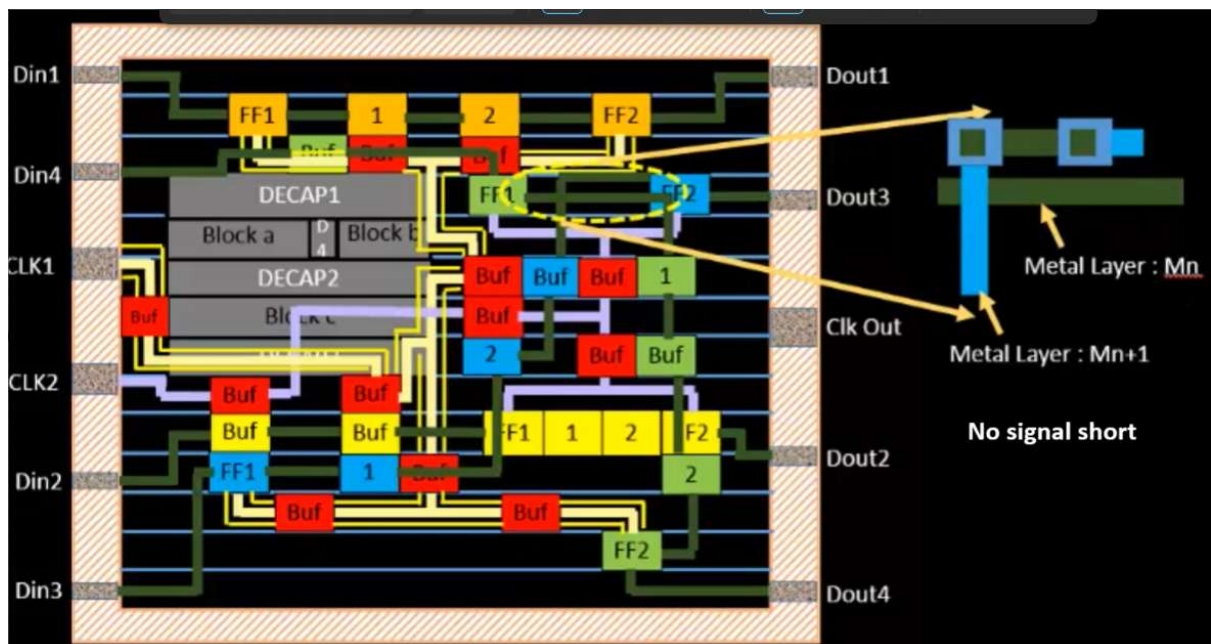
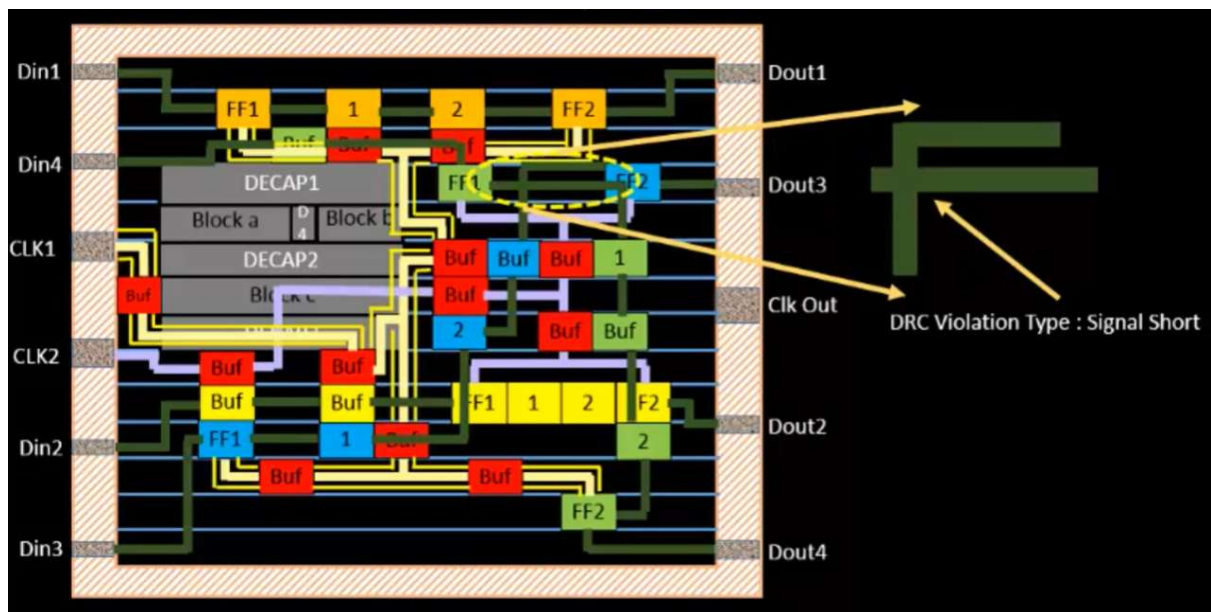


Design Rule Check

Routing doesn't just connect two points; it must follow specific rules, such as maintaining a minimum distance, width, and pitch between wires. DRC (Design Rule Checking) is done to ensure the routes can be properly fabricated and printed on silicon.



Signal shorts, which cause functionality failure, can be avoided by moving the route to a different layer. This introduces more DRC checks, such as via width, spacing, and ensuring higher metal layers are wider than lower ones.

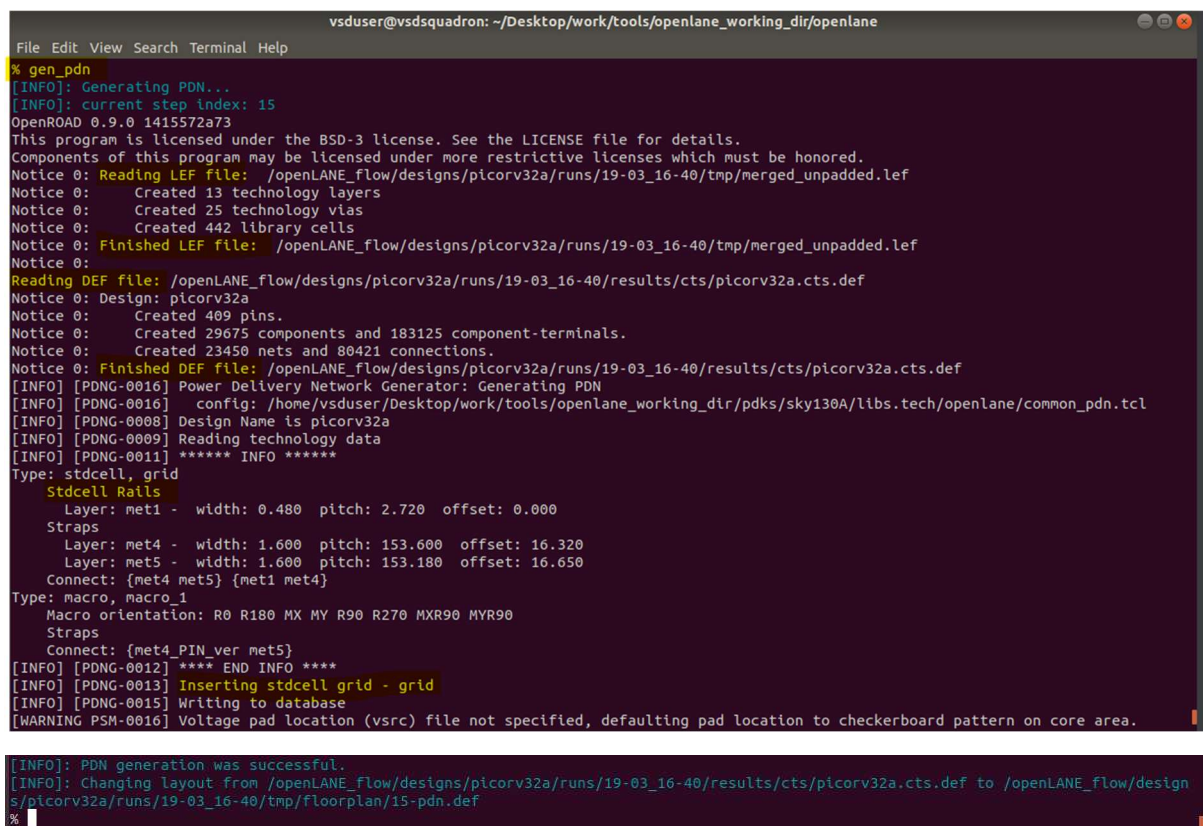


Power Distribution Network and Routing

Lab Steps To Build Power Distribution Network

The lab steps to build a power distribution network are -:

1. Go to the openlane directory
2. Enter the command `docker` and then `./flow.tcl -interactive`. To subsequently get the openlane package, type ***package require openlane 0.9***.
3. Then prep the design using -: `**prep -design picorv32a -tag [folder name of run where in cts had been done]`
4. Then type **`echo $::env(CURRENT_DEF)`**
`/openLANE_flow/designs/picorv32a/runs/[folder name of run where in cts had been done]/results/cts/picorv32a.cts.def`
5. Then, type this to generate the PDN -: **`gen_pdn`**



```
vsduser@vdsquadron: ~/Desktop/work/tools/openlane_working_dir/openlane
File Edit View Search Terminal Help
% gen_pdn
[INFO]: Generating PDN...
[INFO]: current step index: 15
OpenROAD 0.9.0 1415572a73
This program is licensed under the BSD-3 license. See the LICENSE file for details.
Components of this program may be licensed under more restrictive licenses which must be honored.
Notice 0: Reading LEF file: /openLANE_flow/designs/picorv32a/runs/19-03_16-40/tmp/merged_unpadded.lef
Notice 0: Created 13 technology layers
Notice 0: Created 25 technology vias
Notice 0: Created 442 library cells
Notice 0: Finished LEF file: /openLANE_flow/designs/picorv32a/runs/19-03_16-40/tmp/merged_unpadded.lef
Notice 0:
Reading DEF file: /openLANE_flow/designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.def
Notice 0: Design: picorv32a
Notice 0: Created 409 pins.
Notice 0: Created 29675 components and 183125 component-terminals.
Notice 0: Created 23450 nets and 80421 connections.
Notice 0: Finished DEF file: /openLANE_flow/designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.def
[INFO] [PDNG-0016] Power Delivery Network Generator: Generating PDN
[INFO] [PDNG-0016] config: /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech/openlane/common_pdn.tcl
[INFO] [PDNG-0008] Design Name is picorv32a
[INFO] [PDNG-0009] Reading technology data
[INFO] [PDNG-0011] ***** INFO *****
Type: stdcell, grid
Stdcell Rails
Layer: met1 - width: 0.480 pitch: 2.720 offset: 0.000
Straps
Layer: met4 - width: 1.600 pitch: 153.600 offset: 16.320
Layer: met5 - width: 1.600 pitch: 153.180 offset: 16.650
Connect: {met4 met5} {met1 met4}
Type: macro, macro_1
Macro orientation: R0 R180 MX MY R90 R270 MXR90 MYR90
Straps
Connect: {met4_PIN_ver met5}
[INFO] [PDNG-0012] **** END INFO ****
[INFO] [PDNG-0013] Inserting stdcell grid - grid
[INFO] [PDNG-0015] Writing to database
[WARNING PSM-0016] Voltage pad location (vsrsc) file not specified, defaulting pad location to checkerboard pattern on core area.

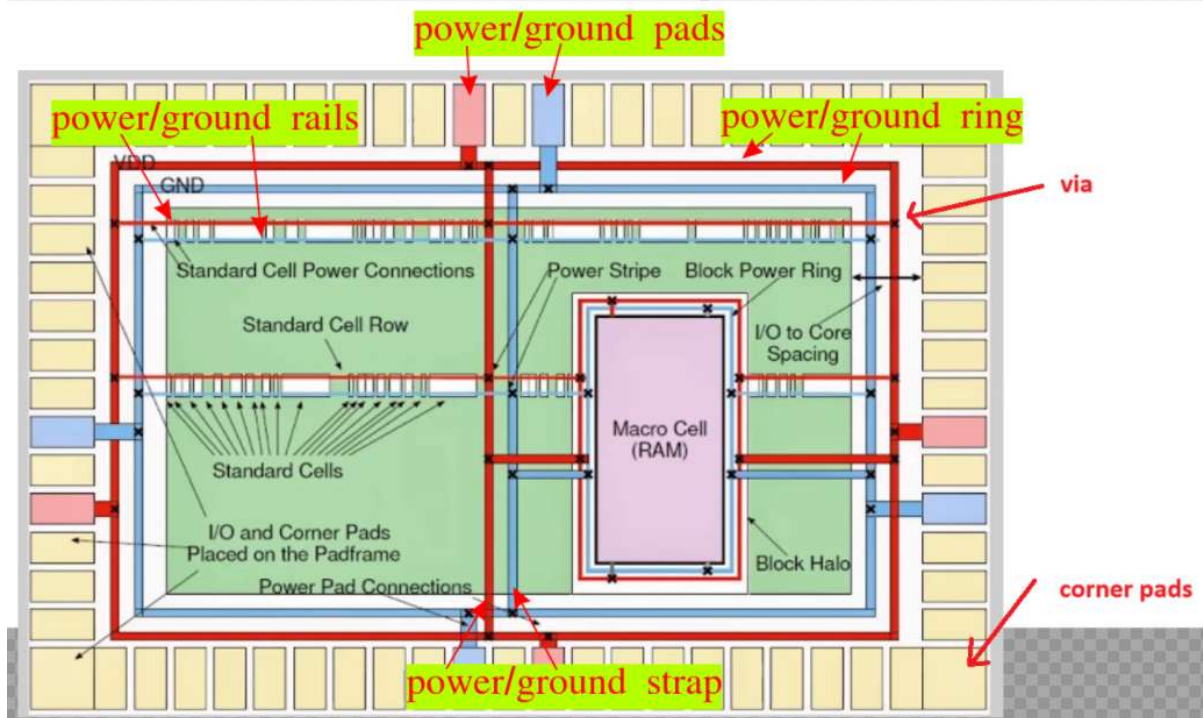
[INFO]: PDN generation was successful.
[INFO]: Changing layout from /openLANE_flow/designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.def to /openLANE_flow/designs/picorv32a/runs/19-03_16-40/tmp/floorplan/15-pdn.def
%
```

Lab Steps From Power Straps To STD Cell Power

The power and ground rails have a pitch of $2.72\mu\text{m}$, so the custom inverter cell also has a height of $2.72\mu\text{m}$ to ensure proper power delivery. In the LEF file (runs/[date]/tmp/merged.lef), all cells have a height of $2.72\mu\text{m}$, with only their width differing.

<pre>MACRO sky130_vsdinv CLASS CORE ; FOREIGN sky130_vsdinv ; ORIGIN 0.000 0.000 ; SIZE 1.380 BY 2.720 ; SITE unithd ; PIN A DIRECTION INPUT ; USE SIGNAL ; ANTENNAGATEAREA 0.165600 ; PORT LAYER l1l ; RECT 0.060 1.180 0.510 1.690 ;</pre>	<pre>MACRO sky130_ef_sc_hd_fakediode_2 CLASS CORE SPACER ; FOREIGN sky130_ef_sc_hd_fakediode_2 ; ORIGIN 0.000000 0.000000 ; SIZE 0.920000 BY 2.720000 ; SYMMETRY X Y R90 ; SITE unithd ; PIN DIODE DIRECTION INPUT ; USE SIGNAL ; PORT LAYER l1l ;</pre>	<pre>MACRO sky130_fd_sc_hd_nand4b_2 CLASS CORE ; FOREIGN sky130_fd_sc_hd_nand4b_2 ; ORIGIN 0.000000 0.000000 ; SIZE 5.520000 BY 2.720000 ; SYMMETRY X Y R90 ; SITE unithd ; PIN A N ANTENNAGATEAREA 0.126000 ; DIRECTION INPUT ; USE SIGNAL ; PORT LAYER l1l ;</pre>
--	--	--

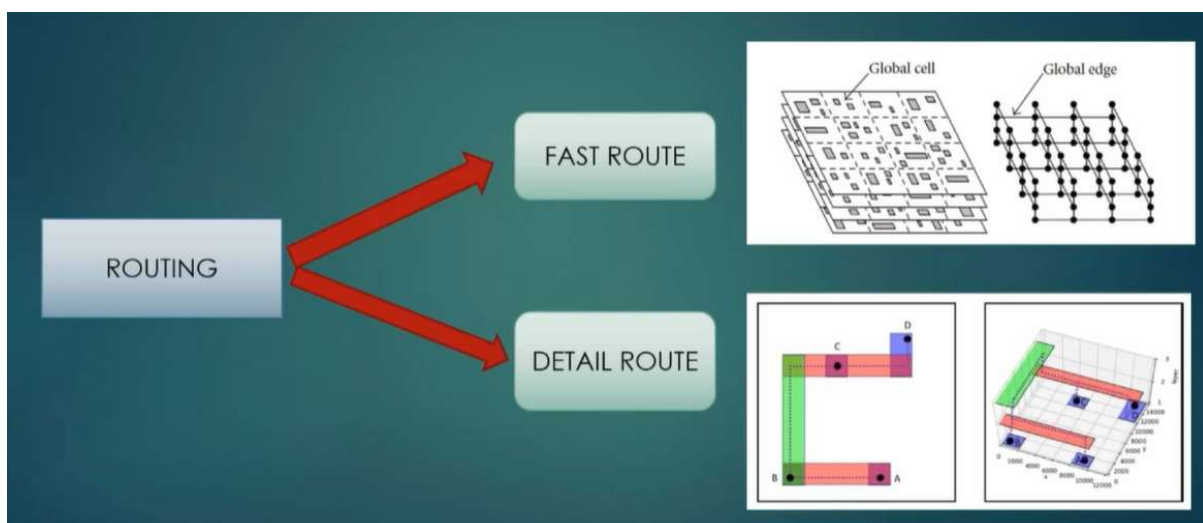
Standard cells are powered up as follows: power/ground pads → power/ground ring → power/ground straps → power/ground rails.



```
% echo $::env(CURRENT_DEF)
/openLANE_flow/designs/picorv32a/runs/19-03_16-40/tmp/floorplan/15-pdn.def
%
```

Basics of Global and Detail Routing and Configure TritonRoute

TritonRoute is the engine that is used for routing through the **run_routing** command.



TritonRoute

- ▶ Performs initial detail route.
- ▶ Honors the **preprocessed route guides** (obtained after fast route) ,i.e. , attempts as much as possible to route within route guides.
- ▶ Assumes route guides for each net satisfy **inter-guide connectivity**.
- ▶ Works on proposed MILP-based **panel routing** scheme with **intra-layer parallel** and **inter-layer sequential** routing framework.

In the VLSI flow, routing is critical and can be done using open-source or commercial tools. It has two phases:

Global Route / Fast Route:

Fast routing techniques partition the area into tiles or rectangles. **Global Routing** sets the initial path framework. **Detail Routing**.

Detail Routing precisely finalizes paths, ensuring proper connectivity and compliance with design constraints.

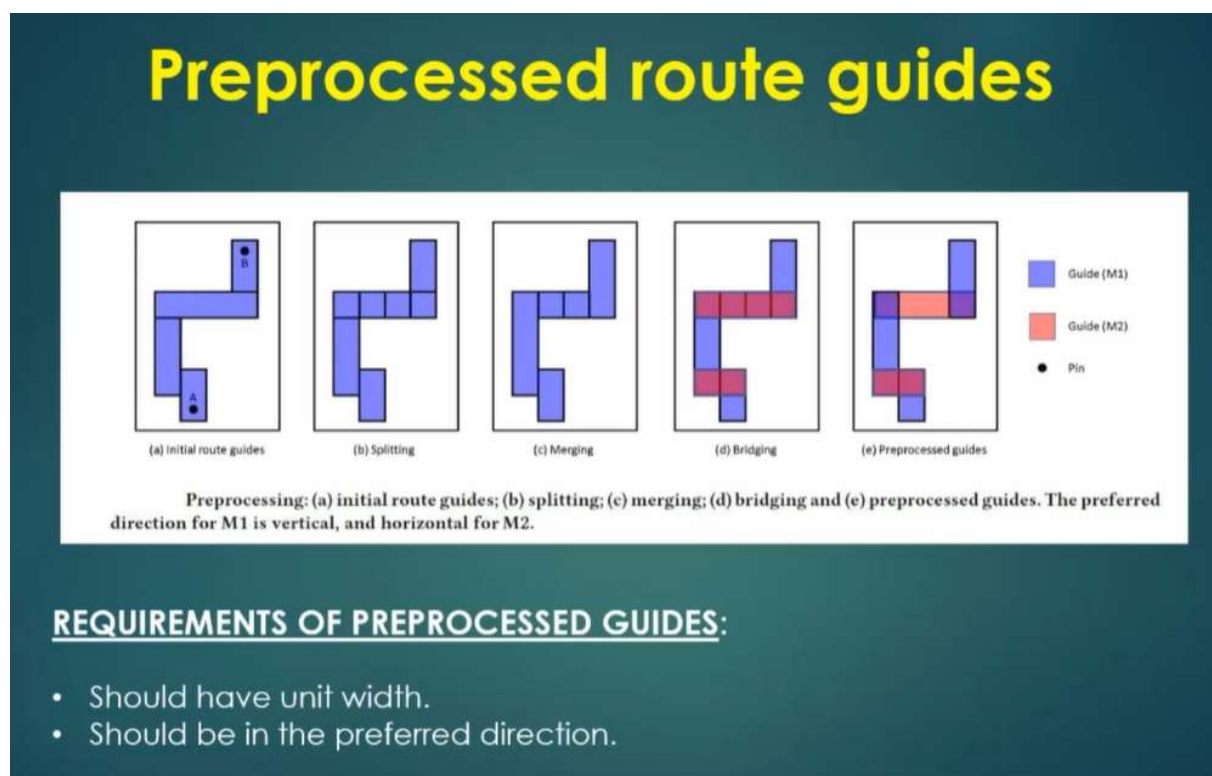
In VLSI, routing is crucial and done using open-source or commercial tools. It has two main phases:

- **Global Route** – Also called fast routing, it partitions the area into tiles/rectangles to establish the initial routing framework.
- **Detailed Route** – Uses precise routing techniques to finalize paths, ensuring connectivity and DRC compliance.

TritonRoute Features

TritonRoute Feature 1 - Honors Pre-processed Route Guides

M1 and M2 prefer vertical and horizontal routing, respectively. If a tool finds a non-preferred route, it splits it into unit widths. Preferred sections are combined, parallel sections are bridged with an upper layer, and non-preferred routes are converted into M2 guides.



TritonRoute Feature 2 & 3 - Inter-Guide Connectivity and Intra & Inter Layer Routing

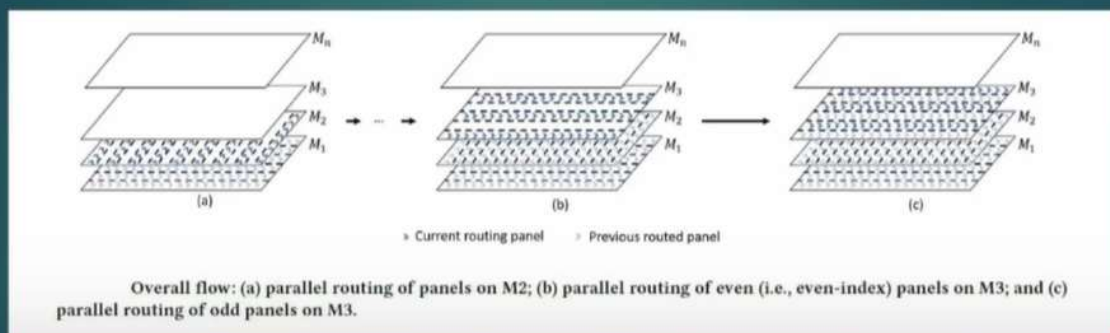
The second feature is interguide connectivity:-

INTERGUIDE CONECTIVITY

- Two guides are connected if :
 - They are on the same layer with edges touching
 - They are on neighbouring metal layers with a nonzero vertical overlap.
- Each unconnected terminal's pin shape must be overlapped by a route guide.

The preferred direction of M1 is vertical, resulting in vertically oriented lines. Panels are dashed lines with routing guides. **Intra-layer panel routing** occurs within even-index panels first, followed by odd-index panels. Routing progresses layer by layer, from lower to upper layers, ensuring an orderly flow.

Intra-layer parallel & Inter-layer sequential panel routing

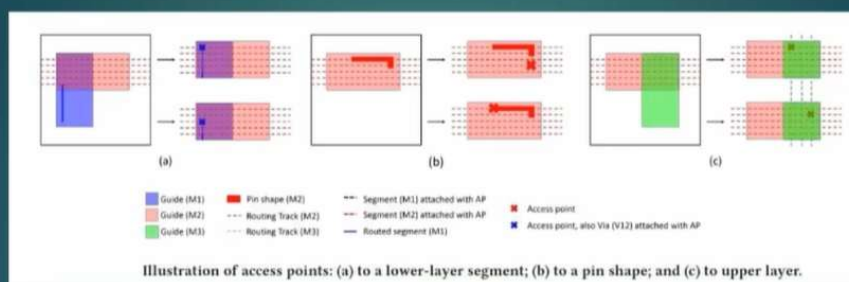


TritonRoute Method To Handle Connectivity

Problem Statement:

- **INPUTS:** LEF, DEF, Preprocessed route guides
- **OUTPUT:** Detailed routing solution with optimized wire-length and via count
- **CONSTRAINTS:** Route guide honoring, connectivity constraints and design rules

► Handling Connectivity



- **Access Point (AP):** An on-grid point on the metal layer of the route guide, and is used to connect to lower-layer segments, upper-layer segments, pins or IO ports.
- **Access Point Cluster (APC):** A union of all Aps derived from same lower-layer segment, upper-layer guide, a pin or an IO port.

The goal of MILP (Mixed Integer Linear Programming) is to find the optimal solution for connecting two access point clusters.

In the algorithm, the cost for each access point is calculated, and a minimum spanning tree is created between access points and their costs. The algorithm aims to find the minimal and most optimal point between two APCs.

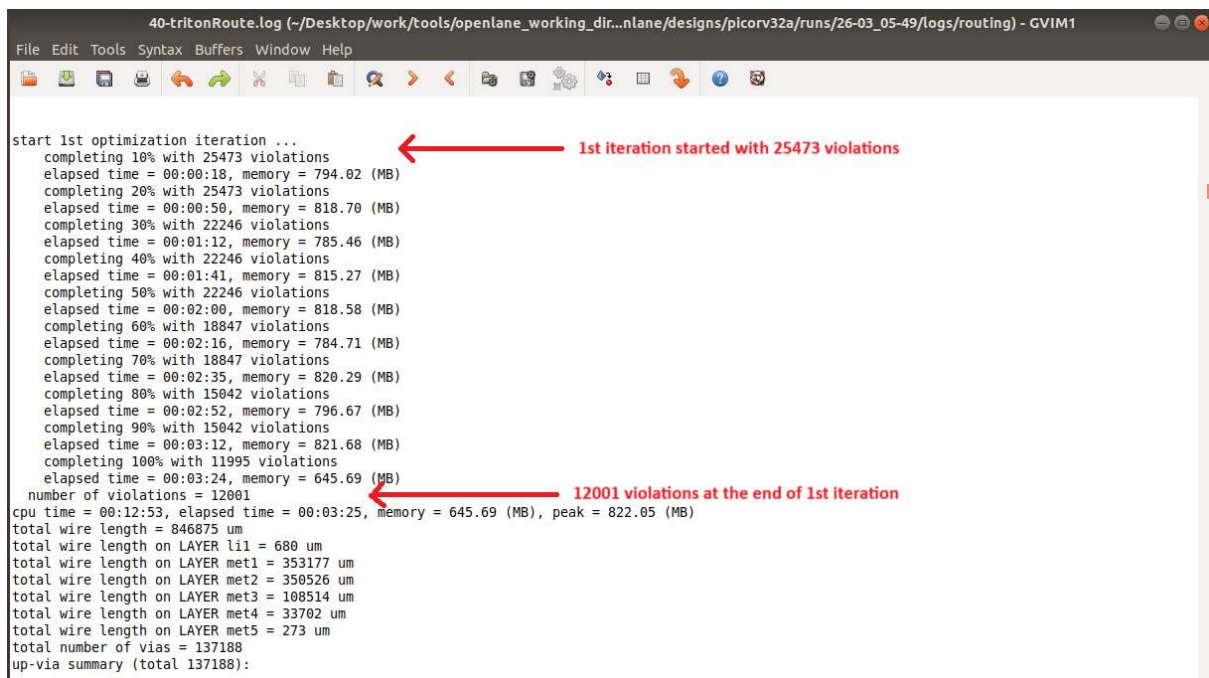
Routing Topology Algorithm

Algorithm 1 Optimization of Routing Topology

```
1: for all  $i = 1$  to  $n - 1$  do
2:   for all  $j = i + 1$  to  $n$  do
3:      $cost_{i,j} \leftarrow dist(APC_i, APC_j)$ 
4:   end for
5: end for
6:  $T \leftarrow MST(APCs, COSTs)$ 
7: Return  $e_{i,j} \in T$ 
```

Routing Topology Algorithm and Final Files List Post Route

After running the run_routing command, both global and detail routing are completed. With a routing strategy set to 0, DRC violations are reduced from 25,000 to 0, which takes about 34 iterations and 20 to 30 minutes.



```
40-tritonRoute.log (~/Desktop/work/tools/openlane_working_dir...nlane/designs/plcorv32a/runs/26-03_05-49/logs/routing) - GVIM1
File Edit Tools Syntax Buffers Window Help
start 1st optimization iteration ...
  completing 10% with 25473 violations
  elapsed time = 00:00:18, memory = 794.02 (MB)
  completing 20% with 25473 violations
  elapsed time = 00:00:50, memory = 818.70 (MB)
  completing 30% with 22246 violations
  elapsed time = 00:01:12, memory = 785.46 (MB)
  completing 40% with 22246 violations
  elapsed time = 00:01:41, memory = 815.27 (MB)
  completing 50% with 22246 violations
  elapsed time = 00:02:00, memory = 818.58 (MB)
  completing 60% with 18847 violations
  elapsed time = 00:02:16, memory = 784.71 (MB)
  completing 70% with 18847 violations
  elapsed time = 00:02:35, memory = 820.29 (MB)
  completing 80% with 15042 violations
  elapsed time = 00:02:52, memory = 796.67 (MB)
  completing 90% with 15042 violations
  elapsed time = 00:03:12, memory = 821.68 (MB)
  completing 100% with 11995 violations
  elapsed time = 00:03:24, memory = 645.69 (MB)
  number of violations = 12001
cpu time = 00:12:53, elapsed time = 00:03:25, memory = 645.69 (MB), peak = 822.05 (MB)
total wire length = 846875 um
total wire length on LAYER li1 = 680 um
total wire length on LAYER met1 = 353177 um
total wire length on LAYER met2 = 350526 um
total wire length on LAYER met3 = 108514 um
total wire length on LAYER met4 = 33702 um
total wire length on LAYER met5 = 273 um
total number of vias = 137188
up-via summary (total 137188):
```

1st iteration started with 25473 violations

12001 violations at the end of 1st iteration

```
40-tritonRoute.log (~/Desktop/work/tools/openlane_working_dir...nlane/designs/picorv32a/runs/26-03_05-49/logs/routing) - GVIM1
File Edit Tools Syntax Buffers Window Help

start 34th optimization iteration ...
  completing 10% with 4 violations ← started with 4 violations
  elapsed time = 00:00:00, memory = 808.90 (MB)
  completing 20% with 4 violations
  elapsed time = 00:00:00, memory = 808.90 (MB)
  completing 30% with 1 violations
  elapsed time = 00:00:01, memory = 808.90 (MB)
  completing 40% with 1 violations
  elapsed time = 00:00:01, memory = 808.90 (MB)
  completing 50% with 1 violations
  elapsed time = 00:00:02, memory = 808.90 (MB)
  completing 60% with 0 violations
  elapsed time = 00:00:02, memory = 808.90 (MB)
  completing 70% with 0 violations
  elapsed time = 00:00:02, memory = 808.90 (MB)
  completing 80% with 0 violations
  elapsed time = 00:00:03, memory = 808.90 (MB)
  completing 90% with 0 violations
  elapsed time = 00:00:03, memory = 808.90 (MB)
  completing 100% with 0 violations
  elapsed time = 00:00:03, memory = 808.90 (MB)
  number of violations = 0 ← 0 violations
cpu time = 00:00:15, elapsed time = 00:00:04, memory = 808.90 (MB), peak = 841.85 (MB)
total wire length = 839218 um
total wire length on LAYER li1 = 562 um
total wire length on LAYER met1 = 311144 um
total wire length on LAYER met2 = 340654 um
total wire length on LAYER met3 = 145248 um
total wire length on LAYER met4 = 41343 um
total wire length on LAYER met5 = 265 um
total number of vias = 139334
up-via summary (total 139334):
```

complete detail routing

total wire length = 839218 um
total wire length on LAYER li1 = 562 um
total wire length on LAYER met1 = 311144 um
total wire length on LAYER met2 = 340654 um
total wire length on LAYER met3 = 145248 um
total wire length on LAYER met4 = 41343 um
total wire length on LAYER met5 = 265 um
total number of vias = 139334
up-via summary (total 139334):

FR_MASTERSLICE		0
li1	57640	
met1	68274	
met2	11487	
met3	1929	
met4	4	

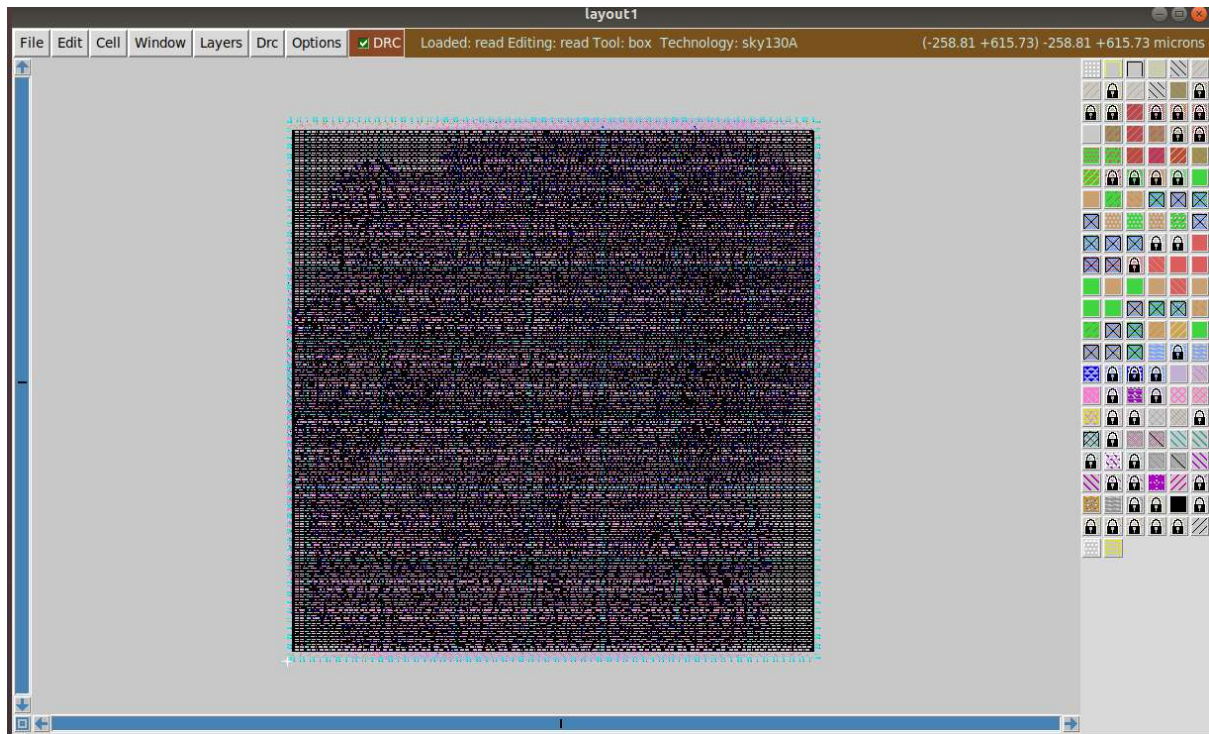
	139334	

cpu time = 01:02:19, elapsed time = 00:17:28, memory = 808.90 (MB), peak = 841.85 (MB)

post processing ...

Runtime taken (hrt): 1059.98

After this, a def file will be formed in the location [runs/[date]/results/routing/picorv32.def], which has to be opened in MAGIC -:



Parasitic Extraction

NOTE: OpenLANE does not have any spef extraction tool, so we use a separate tool present in work/tools/ directory.

The steps for extraction are -:

1. Go to `/home/vsduser/Desktop/work/tools/SPEF_EXTRACTOR`, where you'll find a list of files, including a Python file called `main.py`. This file is used for generating the SPEF. The folder also contains LEF and DEF files.
2. Now, to create the SPEF file `python3 main.py`
`/home/vsduser/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/26-03_05-49/tmp/merged.lef`
`/home/vsduser/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/26-03_05-49/tmp/routing/picorv32a.def`

NOTE: spef will be saved in the same location as def file.

`/home/vsduser/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/26-03_05-49/tmp/routing`

```
vsduser@vdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/26-03_05-49/results/synthesis$ tree -L 1
1
├── merged_unpadded.lef -> ../../tmp/merged_unpadded.lef
├── picorv32a.synthesis_cts.v
├── picorv32a.synthesis_optimized.v
├── picorv32a.synthesis_preroute.v
└── picorv32a.synthesis.v
```

The last stage will be to extract the GDSII file ready for fabrication `run_magic`

This uses Magic to stream the GDSII file `runs/26-03_05-49/results/magic/picorv32a.gds`. This GDSII file can then be read by Magic:

The final stage is to extract the GDSII file for fabrication using `run_magic`. MAGIC streams the GDSII file from `runs/26-03_05-49/results/magic/picorv32a.gds`, making the GDSII ready for fabrication.

```
vsduser@vdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/26-03_05-49/results/magic$ tree -L 1
.
├── merged_unpadded.lef -> ../../tnp/merged_unpadded.lef
├── picorv32a.gds
├── picorv32a.gds.png
├── picorv32a.lef
├── picorv32a.lef.mag
└── picorv32a.mag
```