

# OpenLane Training Manual

## RTL to GDSII Flow

**Engineer. Muhammad Usman**  
**Engineer. Muhammad Shoaib**  
**Dr. Hassan Saif**

## Table of Content

<b>1</b>	<b>INTRODUCTION .....</b>	<b>3</b>
1.1	HOW TO INSTALL OPENLANE .....	3
1.1.1	<i>Installing Dependencies .....</i>	<i>3</i>
1.1.2	<i>Optional Installation.....</i>	<i>5</i>
1.1.3	<i>Installing OpenLane .....</i>	<i>5</i>
<b>2</b>	<b>RTL TO GDSII FLOW.....</b>	<b>6</b>
2.1	STARTING OPENLANE.....	6
2.1.1	<i>Adding New Design to OpenLane .....</i>	<i>6</i>
2.1.2	<i>Synthesis .....</i>	<i>7</i>
2.1.3	<i>Floorplan.....</i>	<i>8</i>
2.1.4	<i>Placement.....</i>	<i>9</i>
2.1.5	<i>Clock Tree Synthesis .....</i>	<i>9</i>
2.1.6	<i>Routing .....</i>	<i>10</i>
2.1.7	<i>Magic (DRC) .....</i>	<i>10</i>
2.1.8	<i>LVS and Antenna Checks .....</i>	<i>11</i>
2.2	NON-INTERACTIVE MODE (METHOD 2).....	11
2.3	USING CARVEL FLOW (METHOD 3) .....	12
<b>3</b>	<b>ASSIGNMENTS: .....</b>	<b>13</b>

## 1 Introduction

OpenLane is a collection of opensource tools that are integrated using OpenLane flow for desinging digital integrated circcuits. This provides an automated RTL to GDS-II flow, using a selection of compenets including OpenROAD, Yosys, Magic, Netgen, Fault, CVC, SPEF-Extractor, CU-GR, Klayout etc. There have also included a number of custom script for design space exploration.

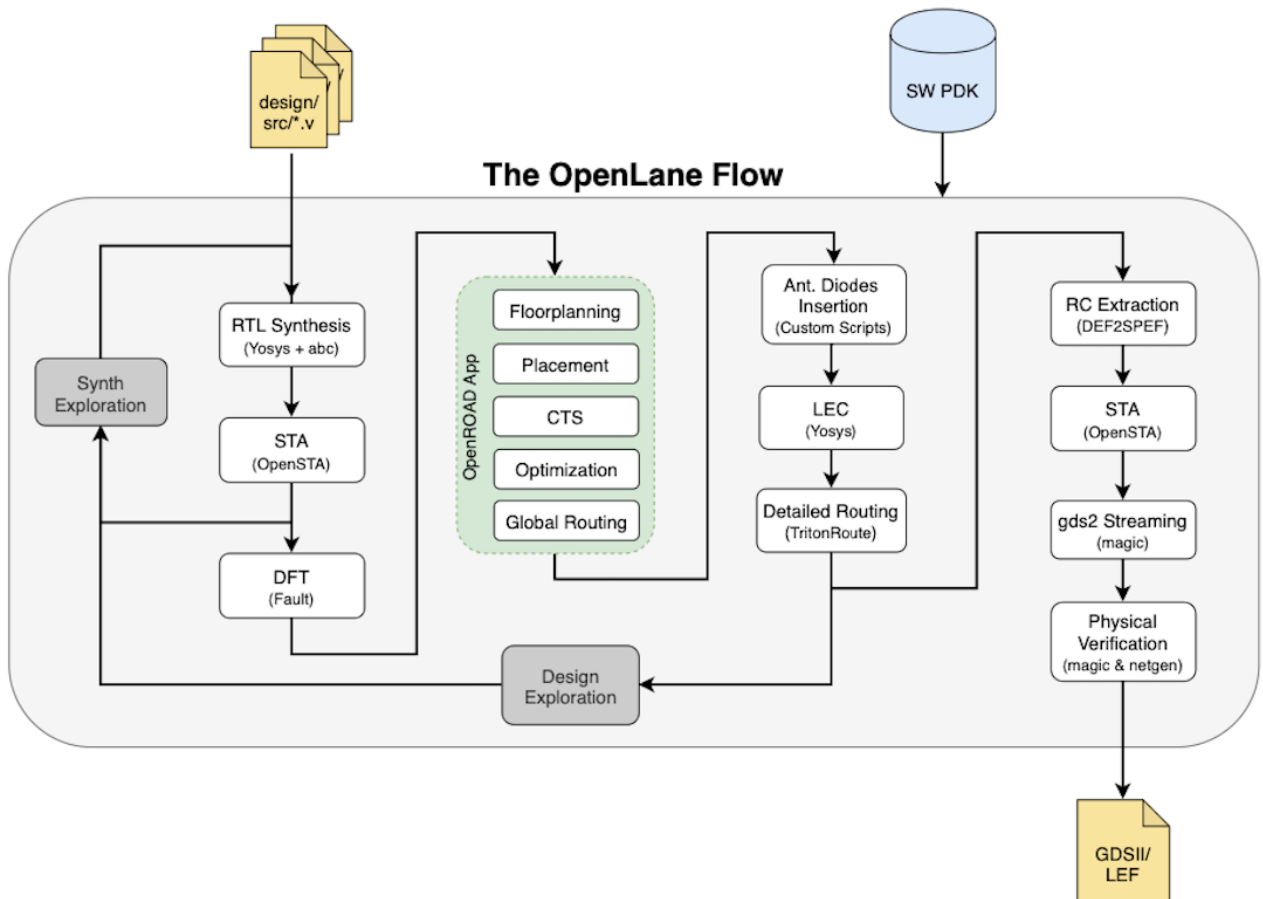


Figure 1.1: OpenLane design flow (source: [https://github.com/The-OpenROAD-Project/OpenLane/blob/master/docs/\\_static/openlane.flow.1.png](https://github.com/The-OpenROAD-Project/OpenLane/blob/master/docs/_static/openlane.flow.1.png))

### 1.1 How to install OpenLane

You need a basic familiralty with linux operating system, we recommend Ubuntu for OpenLane installation. Assuming that you have latest version of the ubuntu installed in your PC or Virtual Machine following are the steps for OpenLane installation.

#### 1.1.1 Installing Dependencies

You can open terminal by right click on an empty space on your screen and then clicking *Open in Terminal* in dropdown menu. Alternatively, you can also use *Ctrl+Alt+T* open terminal. Once terminal is opened run the following command:

```
# Update
sudo apt-get update
```

```
# Install Python3 with pip
sudo apt-get install pip

# Install pyyaml and click
python3 -m pip install pyyaml click

## Install Docker
# Uninstall previous docker images if any
sudo apt-get remove docker docker-engine docker.io containerd runc

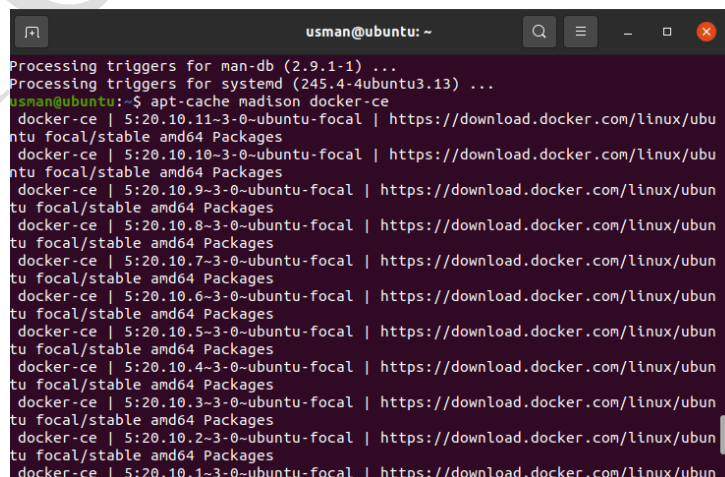
# Install docker
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
apt-cache madison docker-ce
```

The above command will show you a list of docker version you can select one of them and replace the <VERSION\_STRING> and <VERSION\_STRING> in the following command with the output of the above command.



```
usman@ubuntu: ~
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for systemd (245.4-4ubuntu3.13) ...
usman@ubuntu:~$ apt-cache madison docker-ce
docker-ce | 5:20.10.11-3-0-ubuntu-focal | https://download.docker.com/linux/ubu
ntu focal/stable amd64 Packages
docker-ce | 5:20.10.10-3-0-ubuntu-focal | https://download.docker.com/linux/ubu
ntu focal/stable amd64 Packages
docker-ce | 5:20.10.9-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
tu focal/stable amd64 Packages
docker-ce | 5:20.10.8-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
tu focal/stable amd64 Packages
docker-ce | 5:20.10.7-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
tu focal/stable amd64 Packages
docker-ce | 5:20.10.6-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
tu focal/stable amd64 Packages
docker-ce | 5:20.10.5-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
tu focal/stable amd64 Packages
docker-ce | 5:20.10.4-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
tu focal/stable amd64 Packages
docker-ce | 5:20.10.3-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
tu focal/stable amd64 Packages
docker-ce | 5:20.10.2-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
tu focal/stable amd64 Packages
docker-ce | 5:20.10.1-3-0-ubuntu-focal | https://download.docker.com/linux/ubun
```

Figure 1.2: Docker version

```
# Install a specific version using the version string from the second column of the
output from the above command, for example, sudo apt-get install docker-
ce=5:18.09.1~3-0~ubuntu-xenial          docker-ce-cli=5:18.09.1~3-0~ubuntu-xenial
containerd.io
sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=<VERSION_STRING>
containerd.io
```

```
# Verify docker installation
sudo docker run hello-world
```

### 1.1.2 Optional Installation

Optionally you can install Visual Studio Code for editing, or you can also use Linux native editor like, gedit or vi for editing Verilog Code or TCL script etc.

```
# Visual Studio Code
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor >
packages.microsoft.gpg
sudo install -o root -g root -m 644 packages.microsoft.gpg /etc/apt/trusted.gpg.d/
sudo sh -c 'echo "deb [arch=amd64,arm64,armhf signed-
by=/etc/apt/trusted.gpg.d/packages.microsoft.gpg]
https://packages.microsoft.com/repos/code stable main"
/etc/apt/sources.list.d/vscode.list'
rm -f packages.microsoft.gpg
sudo apt install apt-transport-https
sudo apt update
sudo apt install code
```

```
# To install iverilog refer to the following documentation
http://iverilog.icarus.com/
```

```
# To install gtkwave
sudo apt update
sudo apt install gtkwave
```

### 1.1.3 Installing OpenLane

```
## Installing OpenLane
# Go to the directory/folder where you want to install OpenLane and then run the
following commands
git clone https://github.com/The-OpenROAD-Project/OpenLane.git
cd OpenLane/
make pull-openlane
# Following command will install pdk in the openlane directory, of you want to
install pdk elsewhere run the make pdk command followed by export
PDK_ROOT=<absolute path to where skywater-pdk and open_pdk will reside>

make pdk
```

# Following command will install pdk in the openlane directory, if you want to install pdk elsewhere

```
make test
```

Once the result of the command is like shown in the below figure, this shows that you have installed OpenLane and PDK successfully and ready for RTL to GDSII flow using OpenLane.

```
Antenna Summary:
Source: /openlane/designs/spm/runs/openlane_test/reports/routing/38-antenna.rpt
Number of pins violated: 0
Number of nets violated: 0
[INFO]: check full report here: /openlane/designs/spm/runs/openlane_test/reports/final_summary_report.csv
[INFO]: There are no max slew violations in the design at the typical corner.
[INFO]: There are no hold violations in the design at the typical corner.
[INFO]: There are no setup violations in the design at the typical corner.
[SUCCESS]: Flow Completed Without Fatal Errors.
Basic test passed
shahid@shahid-virtual-machine:~/Openlane$
```

Figure 1.3: OpenLane make test results

## 2 RTL to GDSII Flow

### 2.1 Starting OpenLane

To invoke openlane from command line open terminal (you can right click on empty space in gui and then select “Open in terminal” to open terminal window) then, run the following commands:

```
cd <path where you installed OpenLane> (e.g in our case cd Openlane)
make mount
```

it will invoke openlane shell similar to shown in the following figure below

```
shahid@shahid-virtual-machine:~/Openlane$ make mount
cd /home/shahid/Openlane && \
    docker run -it --rm -v /home/shahid/Openlane:/openlane -v /home/
2.34
bash-4.2$
```

Figure 2.1: Openlane Shell

#### 2.1.1 Adding New Design to OpenLane

You can add a new design to designs directory using following command in OpenLane shell:

```
./flow.tcl -design <design_name> -init_design_config
```

It will create following directory/files in the OPENLANE\_ROOT directory

```
designs/<design_name>
├── config.tcl
├── src
└── here your design file should reside
```

Note: you should explore different configuration variable here: <https://github.com/The-OpenROAD-Project/OpenLane/blob/master/configuration/README.md>, these variables can be configured in config.tcl and other library specific configuration files to tune you RTL to GDSII flow.

You can manually copy and paste your design (verilog) files in src directoy. After that run the following command to invoke OpenLane in interactive mode:

```
./flow.tcl -design <design_name> -interactive -tag <tag_name> -overwrite
```

Replace <design\_name> and <tag\_name> with actual design and tag name

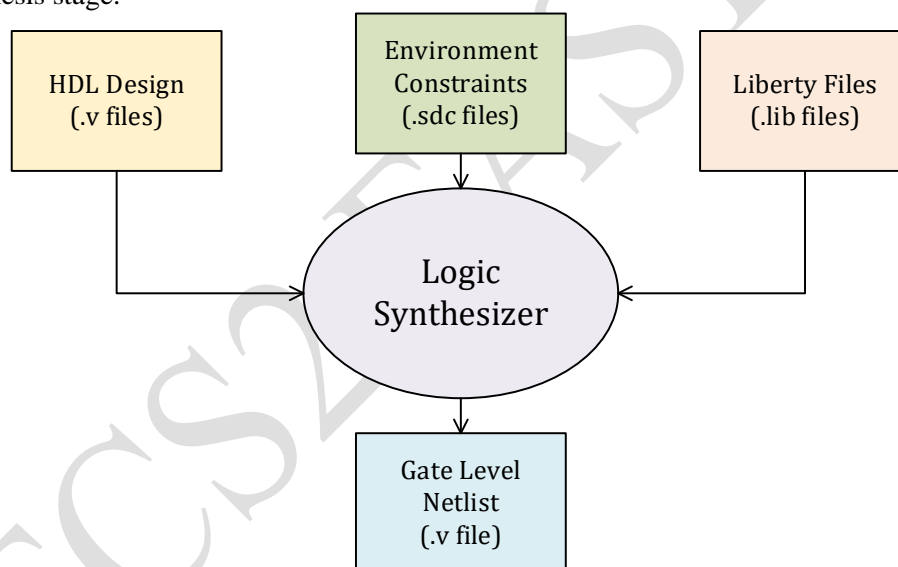
## 2.1.2 Synthesis

Synthesis is a step responsible for converting the RTL logic into technology mapped netlist. Every PDK has some standard cells, onto which our RTL logic is mapped on. In Openlane flow, yosys is responsible for the synthesis and optimization of RTL logic.

This step is executed using follwoing command.

```
run_synthesis
```

This comman synthesizze your design to technology specific standard cells. Following are the input and out put of the synthesis stage.



**Figure 2.2: Synthesis**

Technology library files (.lib) and a standard constrains file are provided with openlane and PDK installation, so we only need our design files here. based on the constraint it also proved static timing analysis reports available in run directory/logs/synthesis directory, gatelevel netlist in expoerted to /results/synthesis directory, while reports are exported to /reports/synthesis directory after successful synthesis.

### 2.1.2.1 Synthesis Exploration

You can explore different synthesis strategy results using `run_synth_exploartion` command this create and html file about the result of the exploration along with logs reports and results in respective run directory.

Best Area	Best Gate Count	Best Delay
153.90	19	983.91
-p	-p	-p

Strategy	Gate Count	Area (um^2)	Delay (ps)	Gates Ratio	Area Ratio	Delay Ratio
S1: -p	26	213.96	988.92	1.368	1.39	1.005
S2: -p	31	242.73	1030.98	1.631	1.577	1.047
S3: -p	31	242.73	1030.98	1.631	1.577	1.047
S4: -p	27	216.46	991.75	1.421	1.406	1.007
S5: -p	19	157.65	1020.07	1	1.024	1.036
S6: -p	19	153.90	983.91	1	1	1
S7: -p	19	157.65	1020.15	1	1.024	1.036

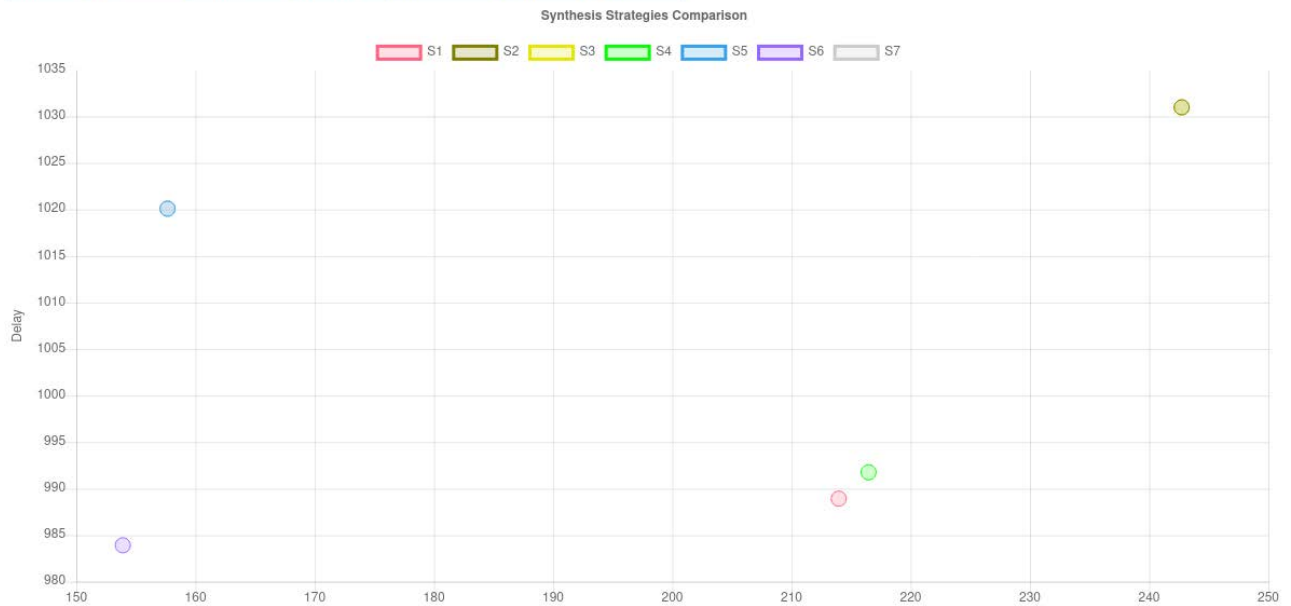


Figure 2.3: Synthesis Exploration Results

### 2.1.3 Floorplan

During the floorplanning, tool takes the above standard cells based netlist and does the power planning of the ASIC design. During the power planning, tool creates the core ring and does special routing of VPWR and VGND to legally place standard cells using abutting. There are different ways to create a floor plan either absolute where you explicitly assign the area where the cells are to be placed or relative floorplane where area is calculated by means of utilization factor.

#### 1 Absolute

```
set ::env(FP_SIZING) "absolute"
set ::env(DIE_AREA) "x0 y0 x1 y1"
```

#### 2 Relative

```
set ::env(FP_CORE_UTIL) "value" (Define in config.tcl file value=0-1)
Die area = cells_area/core_util (e.g  $area = \frac{25\mu m^2}{0.4} = 62.5\mu m^2$ )
```

This step is executed using following command.

```
run_floorplan
```



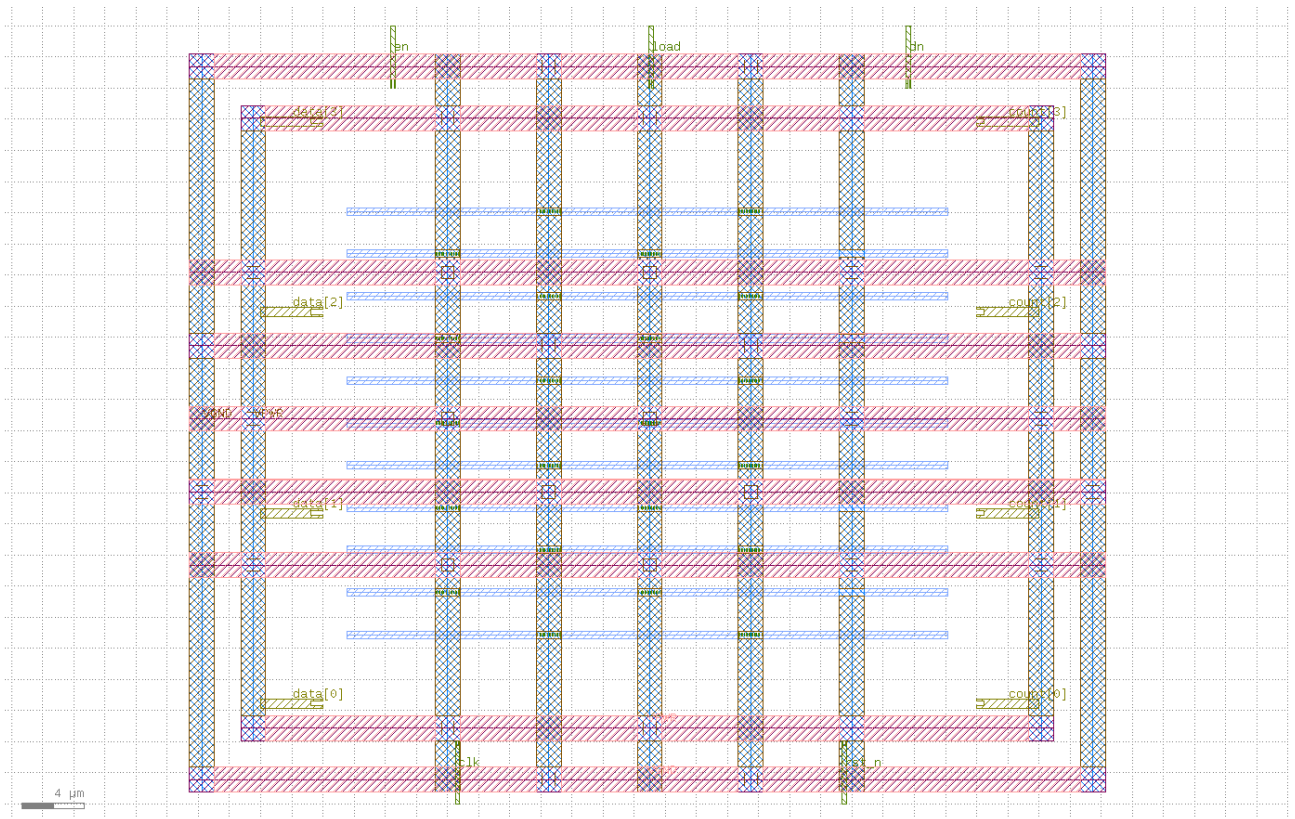


Figure 2.4: Floorplane with power ring

### 2.1.4 Placement

During the placement step, tool places the standard cells into core area and also performs optimization of netlist to take into account the layout information of standard cells. The information related to layout of standard cells is provided in LEF file. Using this information, tool places the standard cells into core area in an optimized manner. The command for this purpose in interactive mode is

```
run_placement
```

placement logs, reports and results can be viewed in respective run directory of you design.

### 2.1.5 Clock Tree Synthesis

Any clock path cannot be treated as any data path. Special cells are provided in standrad cell library for clock signal. Cells used for clock tree synthesis are designed to make sure that they provide equal rise and fall times. Clock signals have high fanout, and to distribute clock evenly among all flipflops clock tree synthesis adds specialized cloc buffer or delay cells.

```
run_cts
```

## 2.1.6 Routing

After the clock tree is formed, in the next step detailed routing step is performed, where tool optimize interconnects and route missing connection make connection with supply and ground and may optimize and replace the cells to achieve targeted performance. At first step global routing is performed after that detailed routing is preformed. In routing toll also look for process antenna violations and try to avoid them and insert antenna diodes where needed to fix antenna violation. Following is the command for the routing in interactive mode.

```
run_routing
```

after routing supply and ground connections are established so you can export powered Verilog. That is Verilog gatelevel netlist with power connections established.

```
write_powered_verilog
```

```
set_netlist $::env(lvs_result_file_tag).powered.v
```

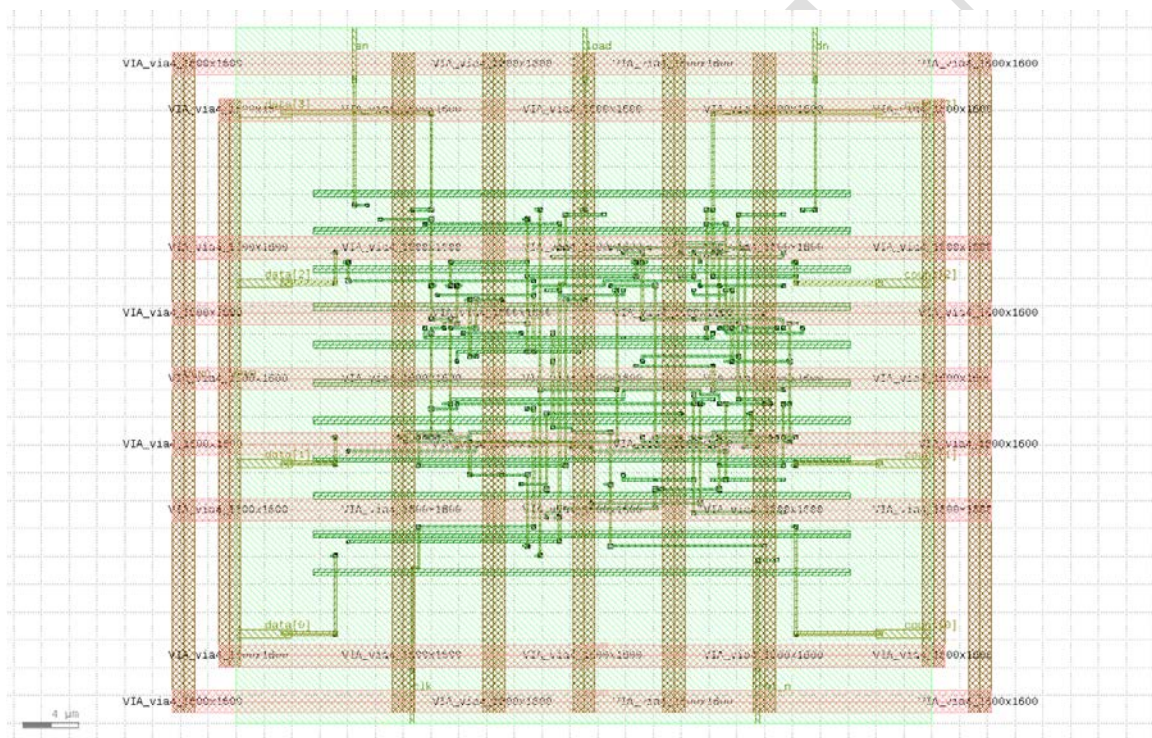


Figure 2.5: Routing

## 2.1.7 Magic (DRC)

Magic is invoked after routing to check design rule violations (DRC), and parasitic extraction using following commands:

```
run_magic
```

```
run_magic_spice_export
```

```
run_magic_drc
```

### 2.1.8 LVS and Antenna Checks

To perform layout vs schematic check powered Verilog gate level netlist is magic extracted spice netlist using following command:

```
run_lvs
```

for antenna check:

```
run_antenna_check
```

Antenna, LVS and DRC reports will be displayed in terminal and will also be exported to respective run directory in the design's directory of the openlane. If all checks are passed along with setup and hold violations, then you have successfully completed the RTL to GDSII flow, otherwise you should explore different configuration as per errors in the config.tcl file (configuration variables: <https://github.com/The-OpenROAD-Project/OpenLane/blob/master/configuration/README.md>)

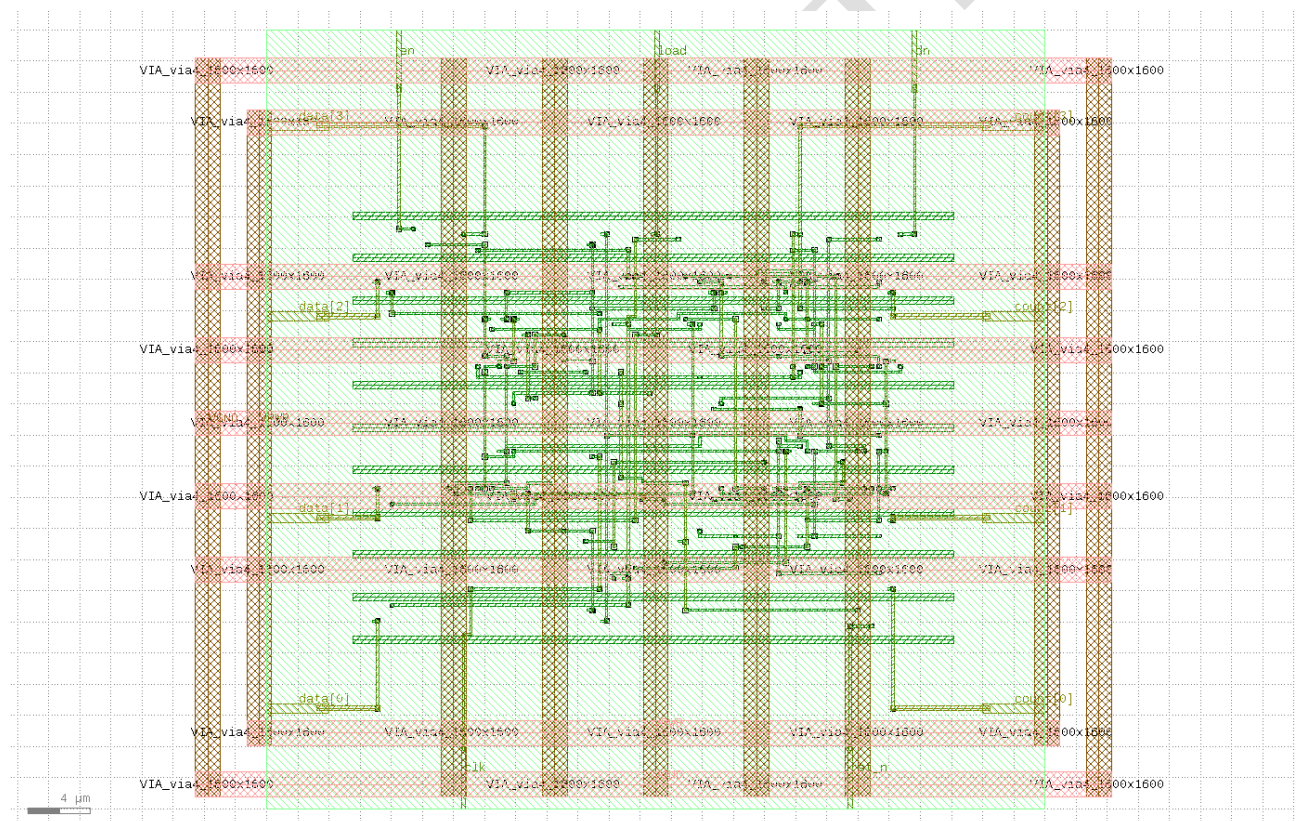


Figure 2.6: Final GDS of counter available at: [https://github.com/muhammadusman7/example\\_codes.git](https://github.com/muhammadusman7/example_codes.git)

## 2.2 Non-Interactive Mode (Method 2)

All the above flow can be completed in single command with non-interactive mode, first you need to set configuration variable in config.tcl files for each step and then run the flow command without interactive switch:

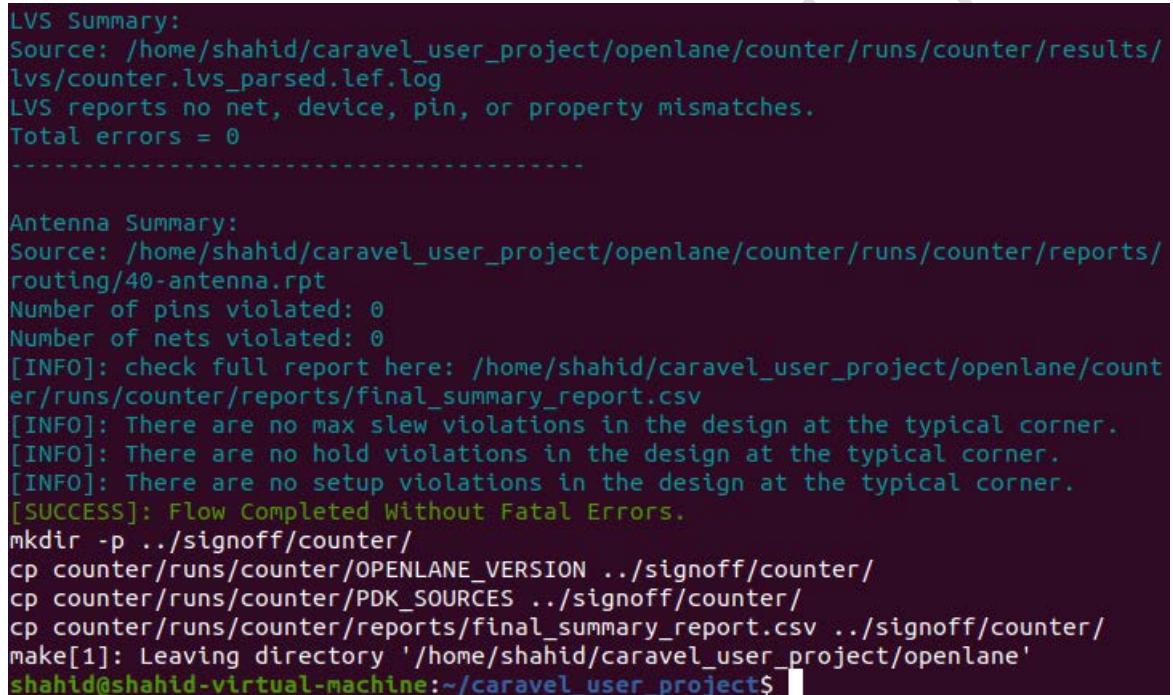


```
./flow.tcl -design counter -tag interactive
```

## 2.3 Using Carvel Flow (Method 3)

Another method to harden your macro is to harden your design using caravel flow (a detailed documentation can be read [here](https://github.com/efabless/caravel_user_project/blob/main/docs/source/quickstart.rst) about installation and running: [https://github.com/efabless/caravel\\_user\\_project/blob/main/docs/source/quickstart.rst](https://github.com/efabless/caravel_user_project/blob/main/docs/source/quickstart.rst)). To complete the flow using caravel create a new folder in CARAVEL\_INSTALLATION/openlane/<design\_name>, there create config.tcl and pin\_order.cfg files (for template of these files you can see the user\_proj\_example directory present in CARAVEL\_INSTALLATION/openlane directory) and add Verilog files in the CARAVEL\_INSTALLATION/verilog/rtl directory. After the initil setup is complete run the following command to complete the flow:

```
make <design_name>
```



```
LVS Summary:
Source: /home/shahid/caravel_user_project/openlane/counter/runs/counter/results/
lvs/counter.lvs_parsed.lef.log
LVS reports no net, device, pin, or property mismatches.
Total errors = 0
-----

Antenna Summary:
Source: /home/shahid/caravel_user_project/openlane/counter/runs/counter/reports/
routing/40-antenna.rpt
Number of pins violated: 0
Number of nets violated: 0
[INFO]: check full report here: /home/shahid/caravel_user_project/openlane/counter/runs/counter/reports/final_summary_report.csv
[INFO]: There are no max slew violations in the design at the typical corner.
[INFO]: There are no hold violations in the design at the typical corner.
[INFO]: There are no setup violations in the design at the typical corner.
[SUCCESS]: Flow Completed Without Fatal Errors.
mkdir -p ../signoff/counter/
cp counter/runs/counter/OPENLANE_VERSION ../signoff/counter/
cp counter/runs/counter/PDK_SOURCES ../signoff/counter/
cp counter/runs/counter/reports/final_summary_report.csv ../signoff/counter/
make[1]: Leaving directory '/home/shahid/caravel_user_project/openlane'
shahid@shahid-virtual-machine:~/caravel_user_project$
```

Figure 2.7: Flow using carvel

### 3 Assignments:

1. Clone the code available at [https://github.com/muhammadusman7/example\\_codes.git](https://github.com/muhammadusman7/example_codes.git) and simulate them using iverilog.
2. Try the counter example with different configurations (apply them in config.tcl file of the design) available at <https://github.com/The-OpenROAD-Project/OpenLane/blob/master/configuration/README.md> and report best configuration
3. Harden alu.v design using caravel flow with absolute floor planning of  $1000\mu m \times 1000\mu m$  and  $500\mu m \times 500\mu m$  and using different placement target densities (hint: use PL\_TARGET\_DENSITY) report the differences (if any)