JOINT INSTITUTE
交大密西根学院

ECE4810J SoC Design

Fall 2024

Lab #4 Getting Started with the Automated ASIC Design Flow based on OpenRoad

Due: 11:59pm Nov.18th, 2024

Logistics:

- This lab is a team exercise.
- Please use the discussion board on Piazza for Q&A.
- All reports and code (if available) MUST be submitted to the EDA platform.
- Internet usage is allowed and encouraged.
- Late penalty (10% per day) will be applied for this lab.

Contents

1 Overview

In this lab, you will learn about ASIC design flow. The goals of this lab are to:

- Learn OpenROAD-flow-scripts.
- Learn about the automated ASIC flow based on the Tencent EDA platform.

2 OpenROAD

2.1 Introduction

The OpenROAD Project was founded in 2018 under the DARPA IDEA program to address the issue of hardware design requiring too much effort, cost, and time. Since then, the project has shown great success in providing a free, open-source implementation for ASIC designs in technology nodes as small as 7nm. Notably, OpenROAD has enabled first-time chip designers, hobbyists, and students to fabricate chips through the Google/Efabless/Skywater 130nm free shuttle program. Over 100 designs have been silicon-tested on the first two shuttle runs, and hundreds more have been taped out [1] [2].

OpenROAD provides a tremendous opportunity for researchers to perform design space exploration, collaborate, and construct real chips in a free and open-source manner. This lab will aim to:

- Introduce researchers to implementation tools, specifically the OpenROAD flow toolchain
- Motivate the use of open-source designs, tools, and platform development kits (PDKs) in research and teaching
- Provide a demonstration of OpenROAD's features and example uses for both computer architecture research and teaching
- Demonstrate a clear path for attendees to turn their RTL designs into silicon through commercial fabs or the free Google/Efabless/Skywater 130nm Shuttle

2.2 Goals

This lab will cover using OpenROAD for both cutting-edge nodes (e.g., ASAP 7nm) and older nodes (e.g., Skywater 130nm).

In this lab, we will present:

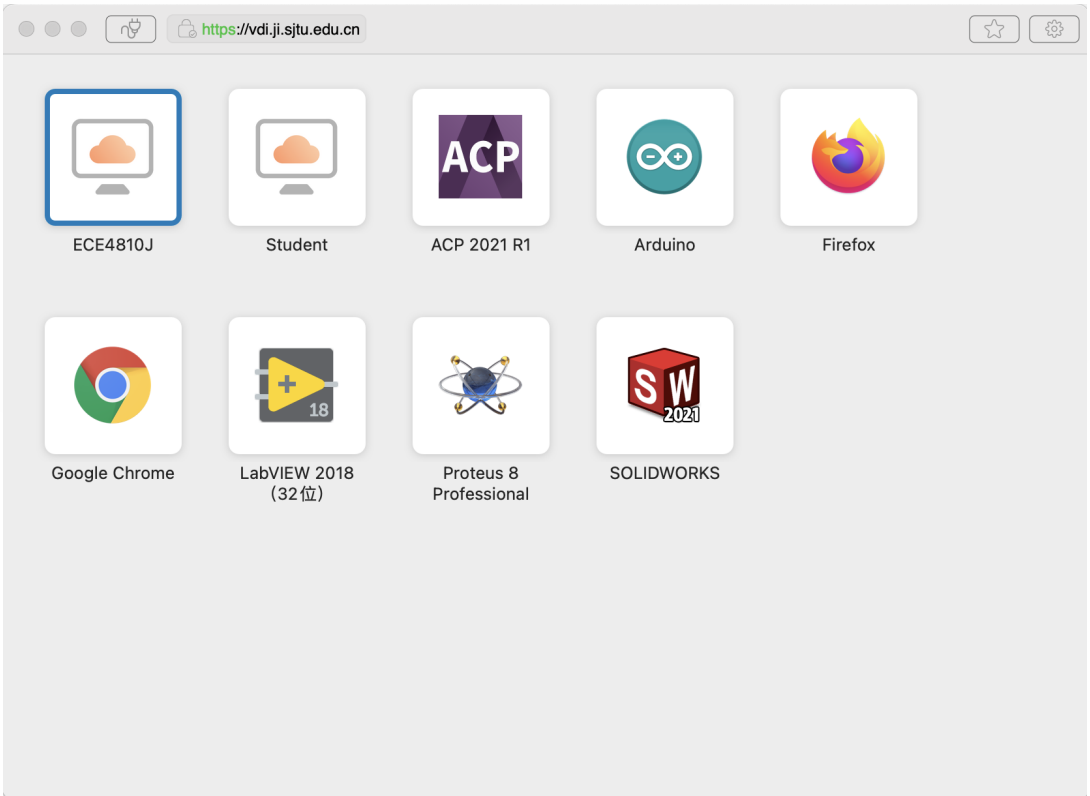
- What is an implementation flow?
  - How does a design get from register-transfer level (RTL) code to a real chip?
  - What is The OpenROAD Project?
  - What are the benefits of open-source hardware flows?
- How do I use OpenROAD?

  - How do I generate a chip from my RTL or example RTL?
  - How do I debug common design problems?
  - How can I collect design data such as power, frequency, and area?
  - How can I model complex designs?
- What are OpenROAD's limitations?
- What is OpenROAD's roadmap?
- How can I contribute to OpenROAD?

This lab is intended to introduce ASIC implementation to those already conceptually familiar with RTL design. We do not cover RTL design techniques or RTL validation.

2.3 EDA Platform Usage

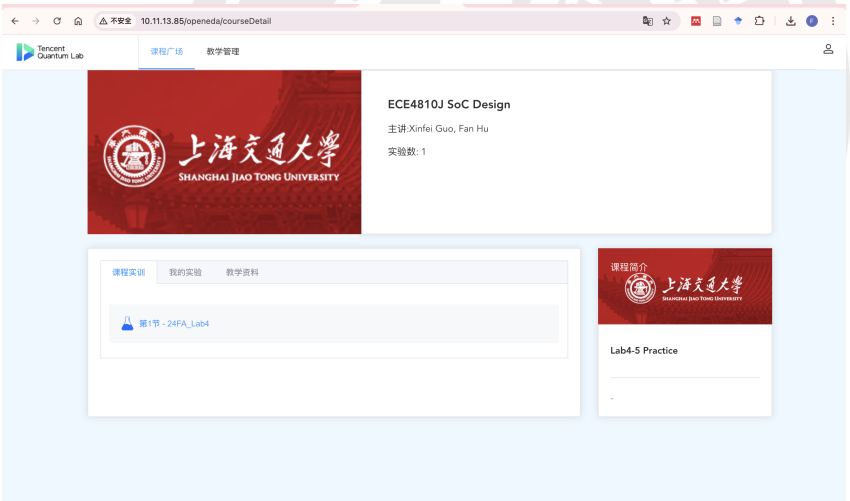
- Visit https://vdi.ji.sjtu.edu.cn (If you are off campus, you need to connect through SJTU VPN).
- Download and install the VMware Horizon Client depend on your own computer system.
- Open VMware Horizon Client and choose the virtual desktop for ECE4810J.



- Run the Google Chrome and visit `http://10.11.13.85/` for the platform. This ip address may change sometimes, so keep an eye out for announcements.

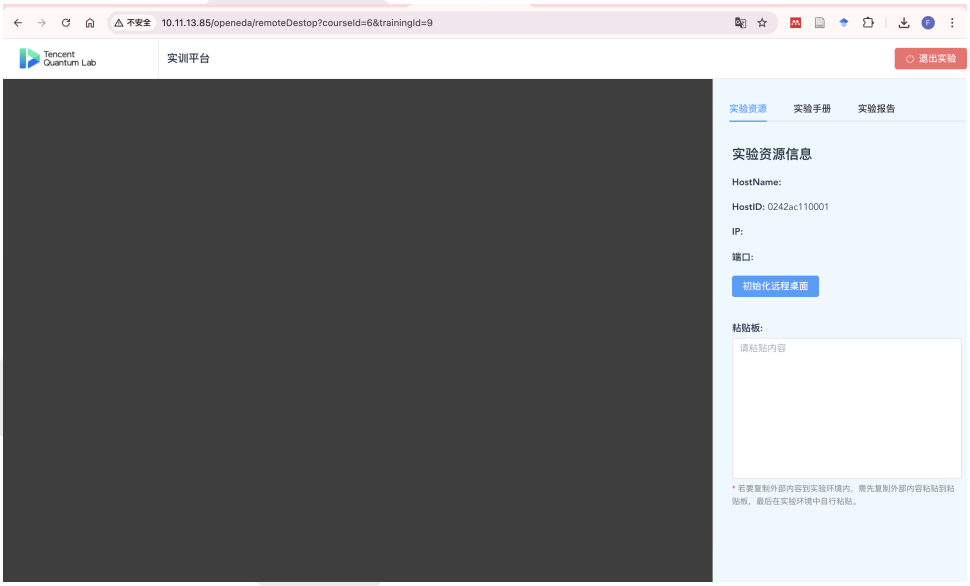
- Click the "Login" button and select the role as "Student".

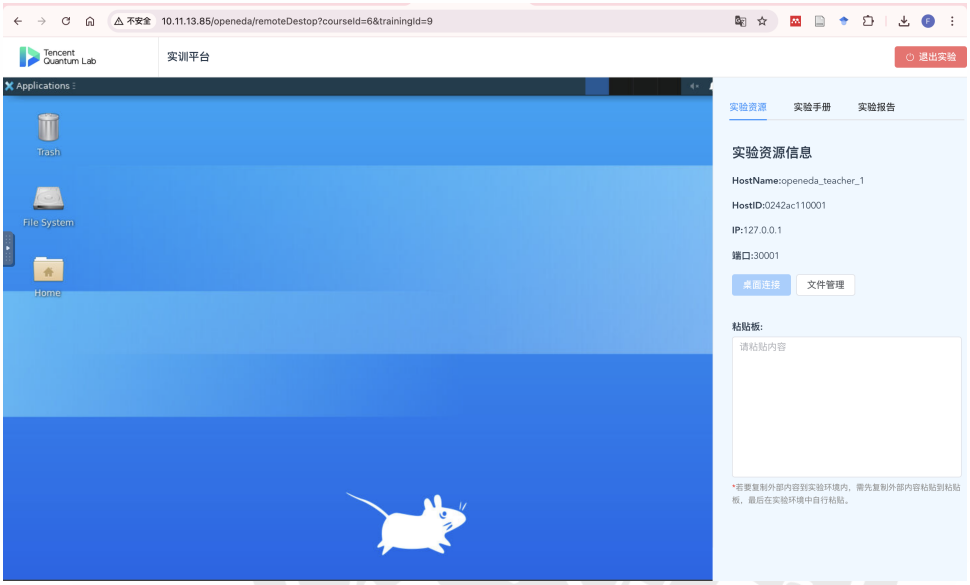
- Enter your account and password and log in. Both of your account and default password are your student number.
- Once you log in for your first time, click the small person icon in the upper right corner and **change your default password**.



- Back to the homepage, click the "Lab 4" tab in the middle. And then click "Initialize remote desktop" on your right to initialize the lab environment for lab 4. If you loss

the connection to the server, click "connect" or "destop connection" should bring you back to the platform.





- Now you can feel free to explore the platform! To open the terminal, click the terminal icon at the bottom of the screen, or use the shortcut **Ctrl**+**Alt**+**T**.
- The platform has preinstalled the tools including openroad, klayout, yosys etc. The directory to the OpenRoad is under `/opt/openeda/tool/OpenROAD-flow-scripts/`. To use the OpenRoad flow scripts, copy the the directory to your own directory and make changes.
- **IMPORTANT:** Only the directory under `/data/experiment` will be permanently stored. Files under other directories may be automatically cleaned after you log out. So you should build all your working directories under this directory.

- Shared public files will be published under `/data/public`. You can copy to your own directory and make changes.
- To upload and download files, click "File Management". To copy and paste words, use the "Paste Board" function.

2.4 Running the Flow

The OpenROAD Project uses three tools to perform automated RTL-to-GDS layout generation:

- yosys: Logic Synthesis
- OpenROAD App: Floorplanning through Detailed Routing
- KLayout: GDS merge, DRC and LVS (public PDKs)

To automate RTL-to-GDS, OpenROAD Flow is provided, which contains scripts that integrate the three tools. The OpenROAD Flow repository serves as an example of RTL-to-GDS flow using the OpenROAD tools. The script build_openroad.sh in the repository will automatically build the OpenROAD toolchain. The two main directories are:

1. `tools/`: contains the source code for the entire `yosys` and OpenROAD App (both via submodules) as well as other tools required for the flow.
2. `flow/`: contains reference recipes and scripts to run designs through the flow. It also contains public platforms and test designs.

Now you are ready to start the lab. To build your own workspace, you should run

```
cd /data/experiment
cp -r /opt/openeda/tool/OpenROAD-flow-scripts/ .
cd OpenROAD-flow-scripts/flow
```

Then you should run `make clean_all` to reset your flow build.

2.4.1 Synthesis

First, to enable yosys usage, you should run:

```
bash /opt/openeda/tool/OpenROAD-flow-scripts/setup_env.sh
source /opt/openeda/tool/oss_cad_suite-0.22.42/oss-cad-suite-env start
```

Run `make synth` and examine the output:

1. Perform file preprocessing (mainly for `yosys`)

```
[INFO][FLOW] Using platform directory ./platforms/nangate45
./util/markDontUse.py -p "TAPCELL_X1 FILLCELL_X1 AOI211_X1
↪  OAI211_X1" -i
↪  platforms/nangate45/lib/NangateOpenCellLibrary_typical.lib -o o
↪  bjects/nangate45/gcd/base/lib/NangateOpenCellLibrary_typical.lib
```

```
Opening file for replace:
↪  platforms/nangate45/lib/NangateOpenCellLibrary_typical.lib
Marked 4 cells as dont_use
Commented 0 lines containing "original_pin"
Replaced malformed functions 0
Writing replaced file: objects/nangate45/gcd/base/lib/NangateOpenCe
↪  llLibrary_typical.lib
mkdir -p ./results/nangate45/gcd/base ./logs/nangate45/gcd/base
↪  ./reports/nangate45/gcd/base
(/usr/bin/time -f 'Elapsed time: %E[h:]min:sec. CPU time: user %U
↪  sys %S (%P). Peak memory: %MKB.'
↪  /OpenROAD-flow-scripts/tools/install/yosys/bin/yosys -v 3 -c
↪  ./scripts/synth.tcl) 2>&1 | tee
↪  ./logs/nangate45/gcd/base/1_1_yosys.log
```

2. Parse input files

```
1. Executing Verilog-2005 frontend: ./designs/src/gcd/gcd.v
2. Executing Liberty frontend.
3. Executing Verilog-2005 frontend:
↪  ./platforms/nangate45/cells_clkgate.v
```

3. Elaborate the design

```
4. Executing SYNTH pass.
4.1. Executing HIERARCHY pass (managing design hierarchy).
4.2. Executing AST frontend in derive mode using pre-parsed AST for
↪  module `\gcd'.
...
4.3. Executing PROC pass (convert processes to netlists).
...
```

4. Optimize the netlist

```
4.4. Executing FLATTEN pass (flatten design).
4.5. Executing OPT_EXPR pass (perform const folding).
4.6. Executing OPT_CLEAN pass (remove unused cells and wires).
4.7. Executing CHECK pass (checking for obvious problems).
4.8. Executing OPT pass (performing simple optimizations).
...
```

5. Map the generic netlist cells to technology-specific cells

```
4.22. Executing TECHMAP pass (map to technology primitives).
...
6. Executing TECHMAP pass (map to technology primitives).
...
9. Executing ABC pass (technology mapping using ABC).
```

2.4.2 Floorplanning

Run `make floorplan` and examine the output:

1. Initialize chip area

   ```
   [INFO IFP-0001] Added 35 rows of 263 sites.
   ```

2. I/O pin placement

   ```
   Using 1u default distance from corners.
   Using 2 tracks default min distance between IO pins.
   [INFO PPL-0007] Random pin placement.
   ```

3. Insert tapcells and endcaps

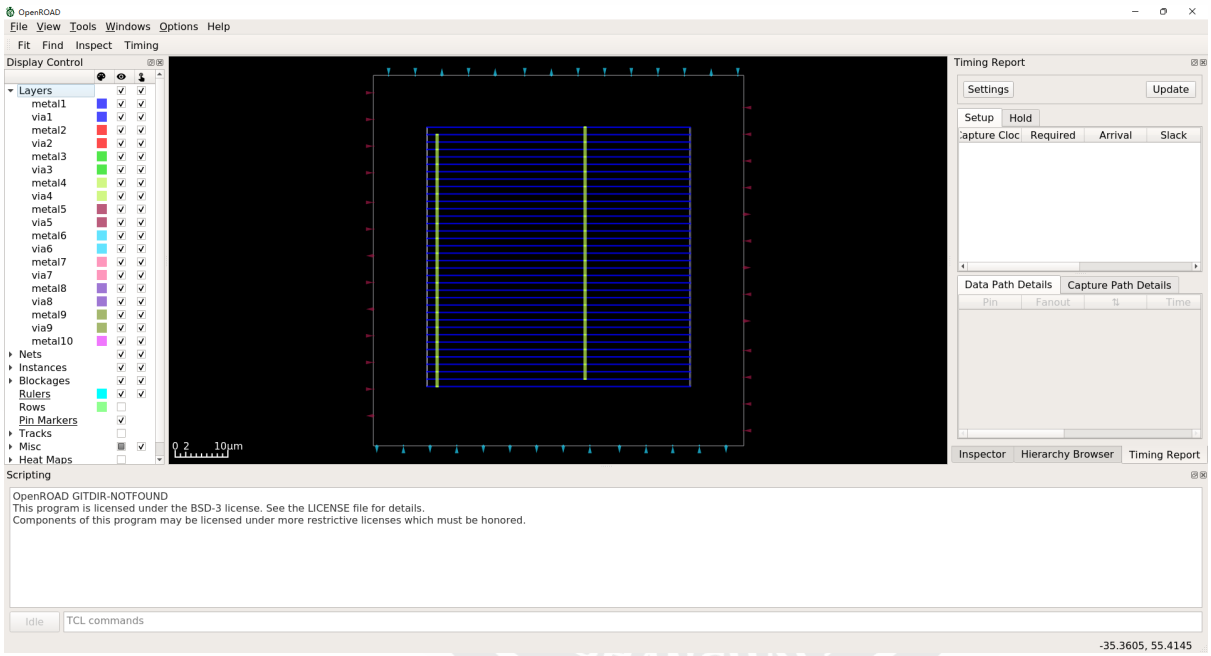
   ```
   [INFO TAP-0004] Inserted 70 endcaps.
   [INFO TAP-0005] Inserted 0 tapcells.
   ```

4. Generate power grid

   ```
   [INFO PDN-0001] Inserting grid: grid
   ```

See this step in the GUI:

`make gui_floorplan`



2.4.3 Global Placement

Run `make place` and examine the output:

1. Initial place

   ```
   [InitialPlace]  Iter: 1 CG residual: 0.00000008 HPWL: 7431482
   [InitialPlace]  Iter: 2 CG residual: 0.00000011 HPWL: 6812877
   ```

```
[InitialPlace]  Iter: 3 CG residual: 0.00000010 HPWL: 6783199
[InitialPlace]  Iter: 4 CG residual: 0.00000010 HPWL: 6764779
[InitialPlace]  Iter: 5 CG residual: 0.00000009 HPWL: 6733515
```

2. Nesterov gradient descent (with timing-driven weighting)

```
[NesterovSolve] Iter: 10 overflow: 0.424423 HPWL: 5509684
[NesterovSolve] Iter: 20 overflow: 0.372513 HPWL: 5236909
[NesterovSolve] Iter: 30 overflow: 0.366609 HPWL: 5245895
[NesterovSolve] Iter: 40 overflow: 0.372344 HPWL: 5230034
[NesterovSolve] Iter: 50 overflow: 0.379915 HPWL: 5258285
...
[INFO GPL-0100] worst slack -6.41e-11
[INFO GPL-0103] Weighted 38 nets.
...
[NesterovSolve] Iter: 340 overflow: 0.108812 HPWL: 5253966
[NesterovSolve] Finished with Overflow: 0.099573
```

3. Timing optimization and electrical rule fixing

```
Perform port buffering...
[INFO RSZ-0027] Inserted 35 input buffers.
[INFO RSZ-0028] Inserted 18 output buffers.
Perform buffer insertion...
[INFO RSZ-0058] Using max wire length 661um.
[INFO RSZ-0039] Resized 39 instances.
Repair tie lo fanout...
Repair tie hi fanout...
```

2.4.4 Detailed Placement

1. Optimize and legalize placement

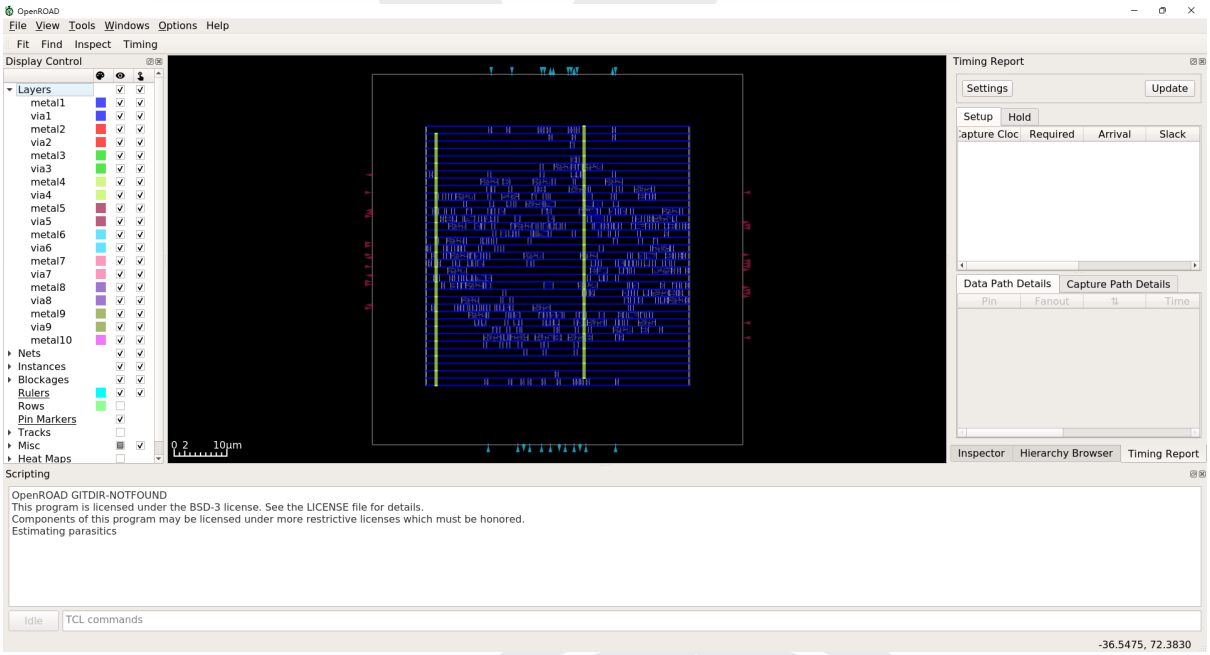
```
Detailed placement improvement.
Importing netlist into detailed improver.
[INFO DPO-0100] Creating network with 470 cells, 54 terminals, 471
↪  edges and 1293 pins.
[INFO DPO-0109] Network stats: inst 524, edges 471, pins 1293
[INFO DPO-0110] Number of regions is 1
[INFO DPO-0401] Setting random seed to 1.
...
Detailed Improvement Results
----------------------------------------
Original HPWL             2885.1 u
Final HPWL                2684.2 u
Delta HPWL                  -7.0 %
```

2. Cell mirroring

```
[INFO DPL-0020] Mirrored 14 instances
[INFO DPL-0021] HPWL before            2684.2 u
[INFO DPL-0022] HPWL after             2681.9 u
[INFO DPL-0023] HPWL delta               -0.1 %
```

See this step in the GUI:

```
make gui_place
```



2.4.5 Clock Tree Synthesis

Run `make cts` and examine the output:

1. Buffer characterization

```
[INFO CTS-0049] Characterization buffer is: BUF_X4.
[INFO CTS-0039] Number of created patterns = 11880.
[INFO CTS-0084] Compiling LUT.
Min. len    Max. len    Min. cap    Max. cap    Min. slew    Max. slew
2           8           1           34          1            14
```

2. Generate the clock tree

```
[INFO CTS-0007] Net "clk" found for clock "core_clock".
[INFO CTS-0010]  Clock net "clk" has 35 sinks.
[INFO CTS-0008] TritonCTS found 1 clock nets.
[INFO CTS-0097] Characterization used 1 buffer(s) types.
[INFO CTS-0027] Generating H-Tree topology for net clk.
[INFO CTS-0028]  Total number of sinks: 35.
```

3. Resize / repair clock tree

```
[INFO RSZ-0058] Using max wire length 661um.
```

4. Legalize buffers

```
Placement Analysis
---------------------------------
total displacement          3.7 u
average displacement        0.0 u
max displacement            1.6 u
original HPWL            2810.3 u
legalized HPWL           2881.1 u
delta HPWL                    3 %
```

5. Repair timing

```
Repair setup and hold violations...
[INFO RSZ-0040] Inserted 3 buffers.
[INFO RSZ-0041] Resized 32 instances.
[WARNING RSZ-0062] Unable to repair all setup violations.
[INFO RSZ-0033] No hold violations found.
```

6. Legalize buffers again

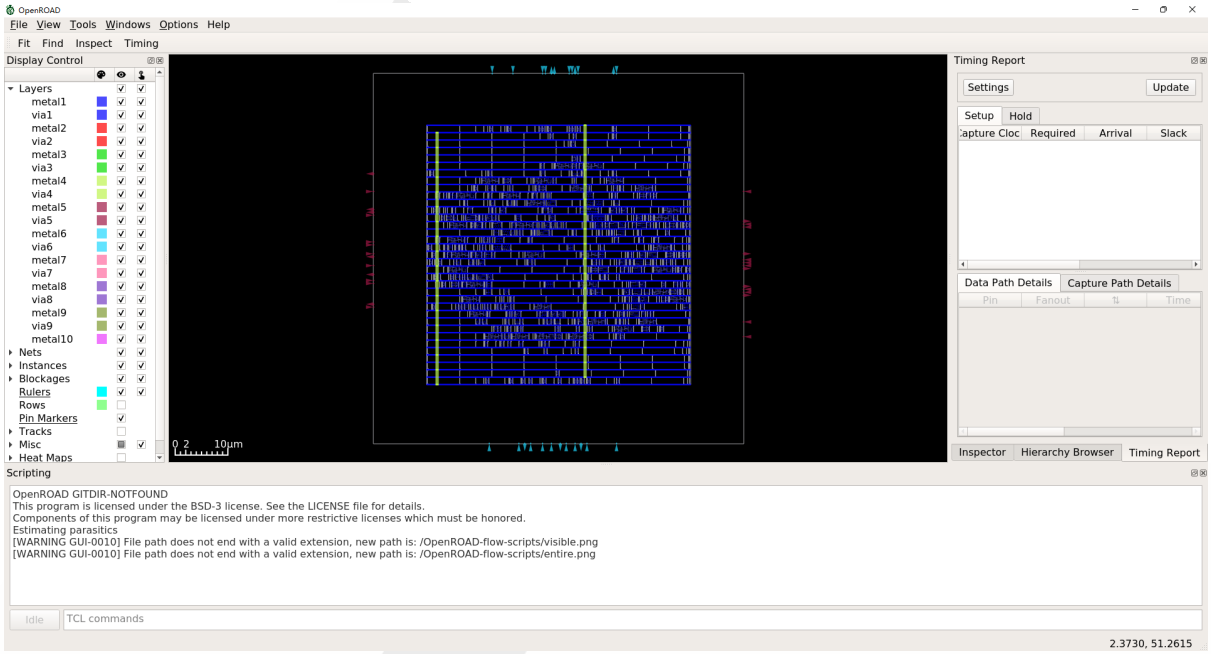
```
Placement Analysis
---------------------------------
total displacement         19.5 u
average displacement        0.0 u
max displacement            2.4 u
original HPWL            2889.4 u
legalized HPWL           2909.1 u
delta HPWL                    1 %
```

7. Insert filler cells

```
[INFO DPL-0001] Placed 734 filler instances.
```

See this step in the GUI:

```
make gui_cts
```



2.4.6 Global Routing

Run `make route` and examine the output:

1. Generate routing grid

```
[INFO GRT-0053] Routing resources analysis:
          Routing      Original      Derated      Resource
Layer     Direction    Resources     Resources    Reduction (%)
-------------------------------------------------------------
metal1     Horizontal          0             0          0.00%
metal2     Vertical        11979          5568         53.52%
metal3     Horizontal      16335          7776         52.40%
metal4     Vertical         7623          5362         29.66%
metal5     Horizontal       7623          5408         29.06%
metal6     Vertical         7623          5408         29.06%
metal7     Horizontal       2178          1120         48.58%
metal8     Vertical         2178          1120         48.58%
metal9     Horizontal       1089          1024          5.97%
metal10    Vertical         1089          1024          5.97%
-------------------------------------------------------------
```

2. Perform global routing

```
[INFO GRT-0096] Final congestion report:
Layer         Resource        Demand        Usage (%)    Max H / Max
 ↪  V / Total Overflow
-------------------------------------------------------------------┘
 ↪  --------------------
```

```
metal1          0          0        0.00%          0
 ↪  /  0 /  0
metal2       5568        743       13.34%          0
 ↪  /  0 /  0
metal3       7776        773        9.94%          0
 ↪  /  0 /  0
metal4       5362         79        1.47%          0
 ↪  /  0 /  0
metal5       5408          4        0.07%          0
 ↪  /  0 /  0
metal6       5408          0        0.00%          0
 ↪  /  0 /  0
metal7       1120          0        0.00%          0
 ↪  /  0 /  0
metal8       1120          0        0.00%          0
 ↪  /  0 /  0
metal9       1024          0        0.00%          0
 ↪  /  0 /  0
metal10      1024          0        0.00%          0
 ↪  /  0 /  0
-------------------------------------------------------------------------
 ↪  --------------------
Total       33810       1599        4.73%          0
 ↪  /  0 /  0
```

3. Check for antenna violations

```
[INFO ANT-0002] Found 0 net violations.
[INFO ANT-0001] Found 0 pin violations.
```

2.4.7 Detailed Routing

1. Region query

```
[INFO DRT-0168] Init region query.
[INFO DRT-0024]   Complete active.
[INFO DRT-0024]   Complete Fr_VIA.
[INFO DRT-0024]   Complete metal1.
[INFO DRT-0024]   Complete via1.
...
```

2. Pin access

```
[INFO DRT-0165] Start pin access.
[INFO DRT-0076]   Complete 100 pins.
[INFO DRT-0078]   Complete 176 pins.
[INFO DRT-0081]   Complete 53 unique inst patterns.
[INFO DRT-0084]   Complete 260 groups.
```

3. Post-process guides

```
[INFO DRT-0169] Post process guides.
[INFO DRT-0176] GCELLGRID X 0 DO 33 STEP 4200 ;
[INFO DRT-0177] GCELLGRID Y 0 DO 33 STEP 4200 ;
[INFO DRT-0028]   Complete active.
[INFO DRT-0028]   Complete Fr_VIA.
[INFO DRT-0028]   Complete metal1.
[INFO DRT-0028]   Complete via1.
```

4. Track assignment

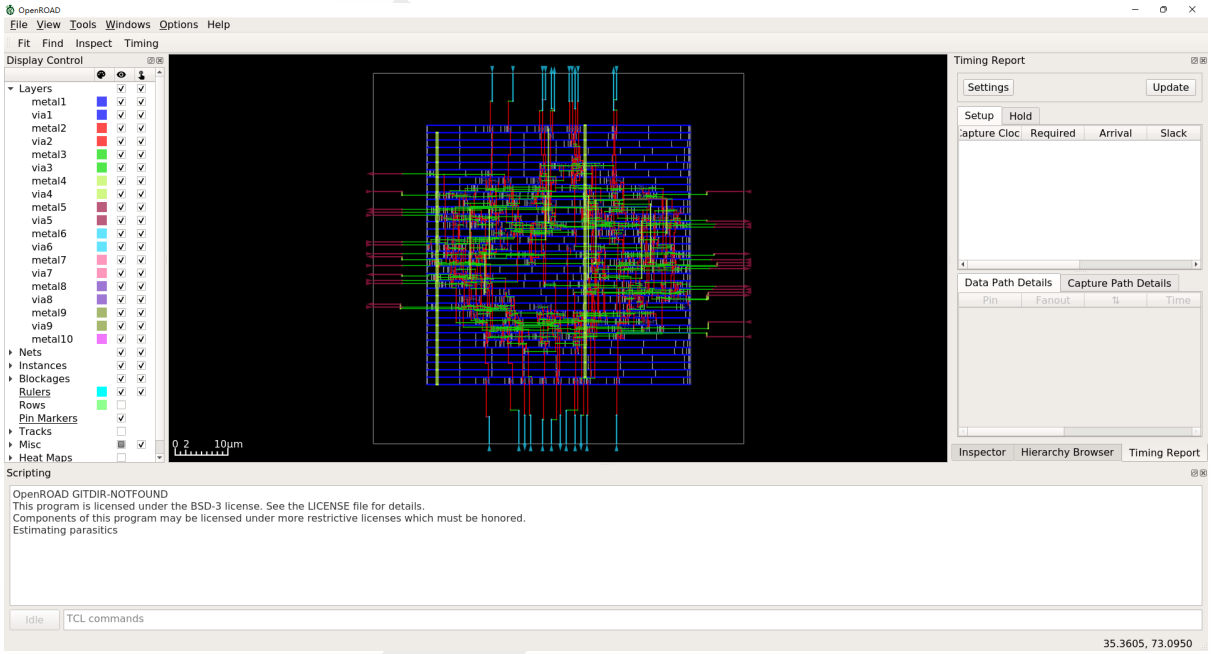
```
[INFO DRT-0181] Start track assignment.
[INFO DRT-0184] Done with 885 vertical wires in 1 frboxes and 1520
↪  horizontal wires in 1 frboxes.
[INFO DRT-0186] Done with 186 vertical wires in 1 frboxes and 316
↪  horizontal wires in 1 frboxes.
[INFO DRT-0182] Complete track assignment.
```

5. Detailed routing

```
[INFO DRT-0194] Start detail routing.
[INFO DRT-0195] Start 0th optimization iteration.
    Completing 10% with 0 violations.
    elapsed time = 00:00:00, memory = 113.23 (MB).
    Completing 20% with 0 violations.
    elapsed time = 00:00:00, memory = 112.47 (MB).
    Completing 30% with 0 violations.
...
    Completing 100% with 31 violations.
    elapsed time = 00:00:01, memory = 130.37 (MB).
[INFO DRT-0199]   Number of violations = 160.
Viol/Layer      metal1   via1 metal2 metal3 metal4 metal5
Cut Spacing           0      1      0      0      0      0
Metal Spacing         2      0      8      0      0      0
Recheck               0      0     87     32      9      1
Short                 1      0     18      1      0      0
...
[INFO DRT-0199]   Number of violations = 0.
...
[INFO DRT-0198] Complete detail routing.
Total wire length = 3568 um.
```

See this step in the GUI:

```
make gui_route
```



2.4.8 Parasitic Extraction

Run `make finish` and examine the output:

Extract parasitic capacitances and resistances

```
[INFO RCX-0008] extracting parasitics of gcd ...
[INFO RCX-0435] Reading extraction model file
 ↪  ./platforms/nangate45/rcx_patterns.rules ...
[INFO RCX-0436] RC segment generation gcd (max_merge_res 50.0) ...
[INFO RCX-0040] Final 1291 rc segments
[INFO RCX-0439] Coupling Cap extraction gcd ...
...
[INFO RCX-0017] Finished writing SPEF ...
```

2.4.9 Timing Signoff

1. Report final timing

```
=========================================================================
 ↪  =======
finish report_tns
-------------------------------------------------------------------------
 ↪  -------
tns -1.50

=========================================================================
 ↪  =======
finish report_wns
```

```
---------------------------------------------------------------------
 ↪  -------
wns -0.09

=====================================================================
 ↪  =======
finish report_worst_slack
---------------------------------------------------------------------
 ↪  -------
worst slack -0.09

=====================================================================
 ↪  =======
finish report_clock_skew
---------------------------------------------------------------------
 ↪  -------
Clock core_clock
Latency      CRPR       Skew
_699_/CK ^
   0.05
_688_/CK ^
   0.05      0.00       0.00
```

2. Report final electrical violations

```
=====================================================================
 ↪  =======
finish max_slew_violation_count
---------------------------------------------------------------------
 ↪  -------
max slew violation count 0

=====================================================================
 ↪  =======
finish max_fanout_violation_count
---------------------------------------------------------------------
 ↪  -------
max fanout violation count 0

=====================================================================
 ↪  =======
finish max_cap_violation_count
---------------------------------------------------------------------
 ↪  -------
max cap violation count 0
```

```
=================================================================
finish setup_violation_count
-----------------------------------------------------------------
setup violation count 0

=================================================================
finish hold_violation_count
-----------------------------------------------------------------
hold violation count 1
```

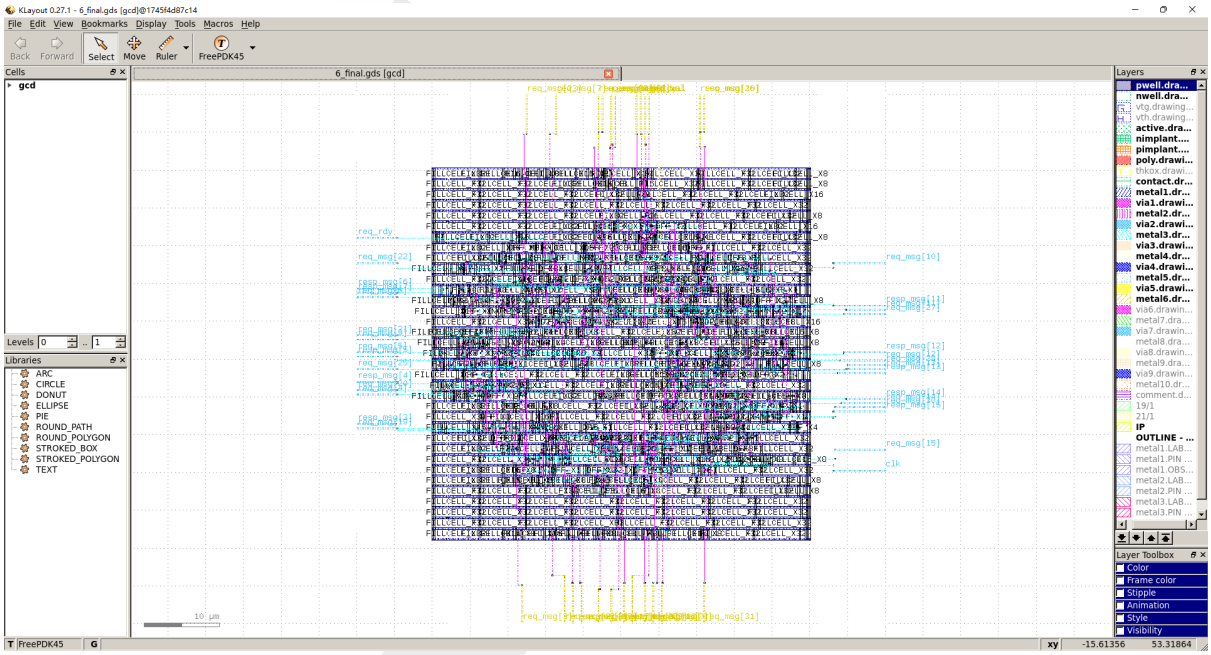
2.4.10 GDS Export

Export DEF file to GDS file

```
[INFO] Reading DEF ...
[INFO] Clearing cells...
[INFO] Merging GDS/OAS files...
       ./platforms/nangate45/gds/NangateOpenCellLibrary.gds
[INFO] Copying toplevel cell 'gcd'
WARNING: no fill config file specified
[INFO] Checking for missing cell from GDS/OAS...
[INFO] Found GDS_ALLOW_EMPTY variable.
[INFO] All LEF cells have matching GDS/OAS cells
[INFO] Checking for orphan cell in the final layout...
[INFO] No orphan cells
[INFO] Writing out GDS/OAS 'results/nangate45/gcd/base/6_1_merged.gds'
```

See this step in the GUI:

```
klayout results/nangate45/gcd/base/6_final.gds
```



2.5 Questions

1. What is half perimeter wire length?
2. What does the Nesterov gradient descent do in global placement?
3. During global placement, the half perimeter wire length, worst slack, and the number of weighted nets when the iteration number of the second Nesterov gradient descent is 320, and the final RC value.
4. What do the global placement and the detailed placement do?
5. What are the names of algorithms that the global placement use? What are the names of algorithms that the detailed placement use?
6. Select one algorithm you list above and briefly introduce the core part of the algorithm.
7. What are the post-pair critical path delay and slack during the clock-tree synthesis?
8. What is the name of the algorithm that the global routing use? What is the name of the algorithm that the detailed routing use?
9. What is parasitic extraction?
10. What are the finish critical path delay and slack during the timing signoff?

2.6 Exercise 1: Debugging a Design #1

Copy files in `/data/public/lab4_starter` to your workspace.

Find the problem with the provided design.

`../../exercise1/config.mk` provides a faulty design config for the design `dynamic_node`, which is a mesh router node. Find the error by running:

```
make DESIGN_CONFIG=../../exercise1/config.mk
```

Once the error is spotted, open `../../exercise1/config.mk` in a text editor and fix the problematic line(s). You can test your solution by cleaning and rerunning the design:

```
# Save time by only cleaning the floorplan step to avoid rerunning synthesis
make DESIGN_CONFIG=../../exercise1/config.mk clean_floorplan
make DESIGN_CONFIG=../../exercise1/config.mk
```

2.7 Exercise 2: Debugging a Design #2

Find the problem with the provided design

`../../exercise2/config.mk` provides a faulty design config. Find the error by running:

```
make DESIGN_CONFIG=../../exercise2/config.mk
```

Once the error is spotted, open `../../exercise2/config.mk` in a text editor and fix the problematic line(s). You can test your solution by cleaning and rerunning the design:

```
# Save time by only cleaning the floorplan step to avoid rerunning synthesis
make DESIGN_CONFIG=../../exercise2/config.mk clean_floorplan
make DESIGN_CONFIG=../../exercise2/config.mk
```

2.7.1 Analyzing Your Design Using OpenROAD

Follow along as the slides demonstrate how to observe design metrics.

This demo will look at the metrics reported for `nangate45/gcd`. If you haven't already, run the design by running make.

Once complete, observe the final report by navigating to `logs/nangate45/gcd/base/6_report.json` for a simple JSON-based report or `logs/nangate45/gcd/base/6_report.log` for a textual report.

2.7.2 Modeling Power

To observe the modeled power, look at `finish__power__total` or `finish report_power`. Note that OpenROAD models power using default activity factors on inputs and propagates these activity factors through the design. This method provides a solid first-order approximation of power and is useful for design space exploration. You can increase the accuracy of the model by applying accurate activity factors on the inputs (see OpenSTA documentation). Static activity-based power modeling (SAIF) and vector-based (VPD) power modeling are even more accurate methods; however, they are not currently supported in OpenROAD.

OpenROAD power report:

```
================================================================================
==
finish report_power
--------------------------------------------------------------------------------
--
Group                  Internal  Switching    Leakage      Total
                         Power      Power      Power      Power (Watts)
----------------------------------------------------------------
Sequential             4.50e-04   5.96e-05   3.16e-06   5.13e-04  31.9%
Combinational          5.65e-04   5.20e-04   9.95e-06   1.10e-03  68.1%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
----------------------------------------------------------------
Total                  1.02e-03   5.80e-04   1.31e-05   1.61e-03 100.0%
                         63.2%      36.0%       0.8%
```

Total power is the most important metric; however, you can read more about the other components here. The power report is broken down by group, where `Sequential` represents flip-flops, `Combinational` represents logic gates, `Macro` represents macros such as SRAM, and `Pad` represents I/O cells (if any).

0.0016

2.7.3 Calculating Max Frequency

To determine the maximum frequency, look at `finish__timing__setup__ws` or finish `report_worst_slack` value. "Slack" is the difference between the constraint (0.46ns) and the actual signal propagation time. Positive slack means the constraint is met ("there is extra slack"). Negative slack means the constraint is violated.

```
================================================================================
==
finish report_worst_slack
--------------------------------------------------------------------------------
--
worst slack -0.09
```

-0.0800

Using the slack, the frequency is calculated as:

$$\text{Frequency}_{\text{max}} = \frac{1}{\text{constraint} - \text{slack}} \tag{1}$$

Be mindful of the sign and units of the slack. Greater slack should mean greater frequency. Be sure that you also calculate frequency using *setup* slack, not *hold* slack.

In this case, the max frequency is:

$$\text{Frequency}_{\text{max}} = \frac{1}{\text{constraint} - \text{slack}} = \ldots \tag{2}$$

2.7.4 Measuring Area

To measure the design area, you must be aware of the different types of the reported area.

1. Synthesized area
   The synthesized area is obtained after synthesis and is a good first-order model for design space exploration. You can find the design area in `reports/nangate45/gcd/base/synth_stat.txt`. Units are μm^2.

   ```
   Chip area for module '\gcd': 519.764000
   ```
   523.754000

2. Place-and-route area
   Place-and-route area is the area obtained after cell placement and routing. If reporting this number, it is implied that the design does not have any violations which make the chip unmanufacturable (e.g., routing or hold time violations). You can find the area from `finish__design__instance__area` or `finish report_design_area`.

   ```
   ===========================================================================
   =======
   finish report_design_area
   ---------------------------------------------------------------------------
   -------
   Design area 584 u^2 24% utilization.
   ```
   593.4460

3. Core area/die area
   Core / Die areas are the most accurate numbers, as they specify the exact area of silicon that will be used for fabrication. However, these numbers are not often reported for computer architecture works. The core area is the area of silicon that cells can occupy. It can effectively be calculated as:

   $$\text{Area}_{\text{core}} = \frac{\text{Area}_{\text{design}}}{\text{utilizaiton}} \tag{3}$$

   The Die area includes all silicon areas needed to fabricate the chip, including any I/O and utilized space.
   In the case of `nangate45/gcd`, the easiest location to find this information is from the design config, which specifies the die area as a set of (x_1, y_1, x_2, y_2) coordinates: flow/designs/nangate45/gcd/config.mk:

   ```
   export DIE_AREA    = 0 0 70.11 70
   export CORE_AREA   = 10.07 11.2 60.04 60.2
   ```

   yielding a die area of:

   $$\text{Area}_{\text{die}} = \ldots \tag{4}$$

   and core area of:

   $$\text{Area}_{\text{core}} = \ldots \tag{5}$$

   which can also be obtained from the previous formula:

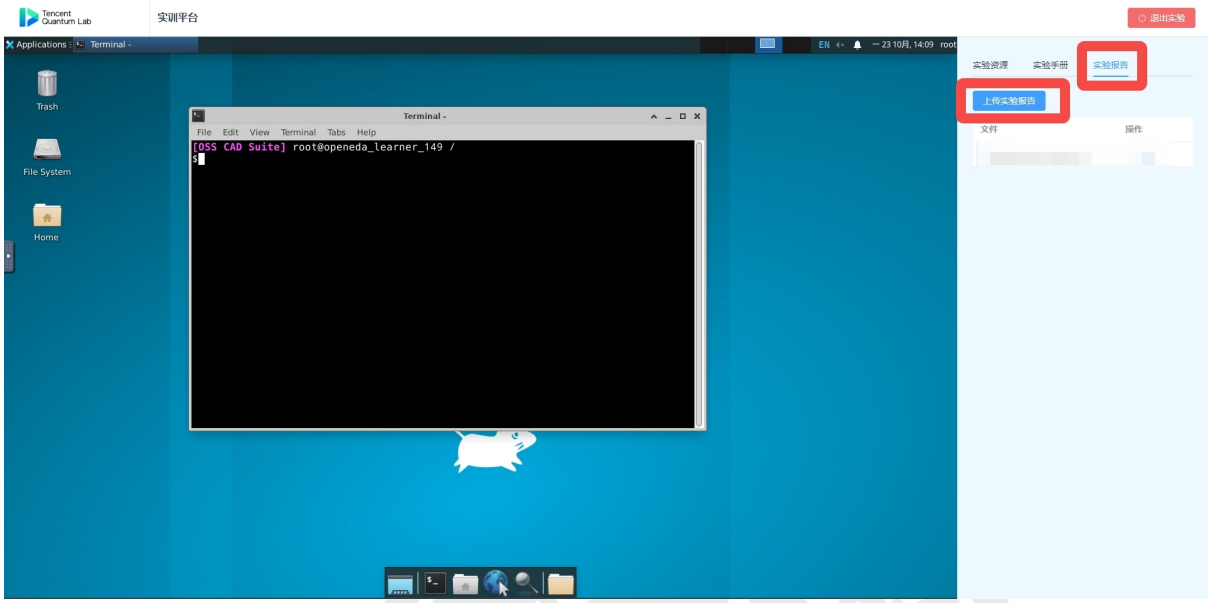
   $$\text{Area}_{\text{core}} = \frac{\text{Area}_{\text{design}}}{\text{utilizaiton}} = \ldots \tag{6}$$

3 Deliverables

Please add comments or explanations to all the deliverables.

1. Section 2.4: the final exported GDS file.
2. Section 2.5: the answers to the questions.
3. Section 2.6: the modified `config.mk` file.
4. Section 2.7: the modified `config.mk` file, calculated 2, 4, 5, and 6.

Note: We will try using the EDA platform for the submission. Please compress all the files into one zip file and upload it through "Experiment Report -> Upload Experiment Report" on your right hand side of the screen.



4 Grading policy

Factors	Percentage
Section 2.4	40%
Section 2.5	20%
Section 2.6	20%
Section 2.7	20%

A Peer Evaluation Form

Part	Your work	Your partner's work	Your score	Your partner's score
Section 2				

B Change Log

Fall 2023: Yichen Cai

- moved Lab 6 to Lab 4
- created instructions for the EDA platform
- updated Section 2

Fall 2022: Yihua Liu

- created this lab

References

[1] T Ajayi et al. "OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain". In: *Proc. GOMACTECH* (2019), pp. 1105–1110.

[2] Tutu Ajayi et al. "Toward an open-source digital flow: First learnings from the openroad project". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, pp. 1–4.