ECE4810J System-on-Chip (SoC) Design

Fall 2021

Lab #6 Getting Started with the OpenLane ASIC Design Flow and Skywater 130nm Technology

Due: November 21st 11:59pm, 2021

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 assignment of Canvas.
- Internet usage is allowed and encouraged.
- No late submission is allowed for this lab.

1. Overview

The purpose of this lab is to familiarize you with the OpenLane VLSI design flow and the Skywater 130nm PDK. OpenLane is an open-source VLSI flow built around open-source tools. That is, it's a collection of scripts that run these tools, in the right order, transforming their inputs and outputs as appropriate, and organizing the results.

OpenLane and the 130nm PDK are the result of a recent, concerted effort by various industry players to democratize ASIC design. To understand the project, watch the introductory presentations made by various key contributors:

- [FOSSi Dial-Up] Tim Ansell - Skywater PDK: Fully open source manufacturable PDK for a 130nm process
- [FOSSi Dial-Up] Mohamed Shalan - OpenLane, A Digital ASIC Flow for SkyWater 130nm Open PDK

And for interest:

- [FOSSi Dial-Up] Mohamed Kassem - The striVe RISC-V SoC Family on SkyWater 130nm
- [FOSSi Dial-Up] James Stine - Designing new 130nm cells for SkyWater 130nm

2. TCL

For reasons that will become obvious, most digital design flows use or heavily rely on TCL scripts in some way. TCL ("Tool Command Language") is designed specifically to address the need to

run multiple disparate tools and glue them together. It's less complicated than Python and more sane than any shell script.

It's straightforward to read, but a non-trivial grasp of the language will be invaluable in your career as an electronic design engineer (or even as a software engineer).

A *lot* of OpenLane is written in TCL.

Work through the first ~15 sections of the TCL Tutorial. And finish the following exercises (you can run tcl under any Vivado shell or run it online https://www.tutorialspoint.com/execute_tcl_online.php)

Problems:

- Write a procedure (proc) called "avg" that takes an array of numbers with any length and return the average of it.
  For example:

```
> puts [avg {70 80 50 60}]
> 65
```

- Write a procedure (proc) that does the factorial function.

```
> puts [factorial 3]
> 6
```

3. Setting up the OpenLane flow

A key benefit of using an open-source tool flow, though, is that you can run it at home. For this lab we will try to do just that.

3.1 Prerequisite

- Ubuntu 20.04/18.04 with more than 10GB free space
- Docker 19.03.12+ ([Install Docker on Ubuntu] (https://docs.docker.com/engine/install/ubuntu/))
- GNU Make
- Python 3.6+ and PIP

3.2 Setting Up OpenLane

```
$ python3 -m pip install pyyaml click
$ git clone https://github.com/The-OpenROAD-Project/OpenLane.git
$ cd OpenLane/
$ make openlane
$ make pdk
```

Note: If clone failed, please access through VPN and configure git proxy.

```
$ make test
```

The test is based on a 5-min-runtime design `spm`.

3.3 Running OpenLane

```
$ make mount # mount into container
$ ./flow.tcl -design <design folder path>
```

Note: Check `/Openlane/designs/` for all designs. No matter flow succeeded or failed, a folder named after the time stamp of running should be created under the directory `/Openlane/designs/<design folder path>/runs/`. Detailed logs and reports should be recorded in this folder.

For better understanding and advanced operations of OpenLane, you may optionally run Openlane [interactively](https://github.com/The-OpenROAD-Project/OpenLane/blob/master/docs/source/advanced_readme.md).

3.4 Displaying the Final Layout

Download [Klayout](https://github.com/The-OpenROAD-Project/OpenLane#flow-configuration). If flow succeeded, run

```
$ cd <design folder path>/runs/<run folder>/results/magic
$ cd <design folder path>/runs/<run folder>/results/klayout
Then `$ klayout <design name>.gds`
```

3.5 Adding a Design

```
$ make mount
$ ./flow.tcl -design <design_name> -init_design_config
```

The following directory structure will be initialized:

```
Openlane/designs/<design_name>
├── config.tcl
├── src
│   ├──
```

Edit the required configuration variables in `config.tcl` and put your design in `/src`. More configurations can be made by either editing variables in `config.tcl`, or in `$::env(DESIGN_DIR)/$::env(PDK)_$::env(STD_CELL_LIBRARY)_config.tcl`. You are also allowed to add an SDC file to the design, but remember to specify its path in `config.tcl`. An example is shown below:

```
Openlane/designs/your_design
├─── config.tcl
├─── sky130A_sky130_fd_sc_hd_config.tcl
├─── src
│    ├─── your_design.v
│    ├─── your_design.sdc
# specify ::env(SDC_FILE) as [glob $::env(DESIGN_DIR)/src/constraint.sdc]
```

Check https://github.com/The-OpenROAD-Project/OpenLane/blob/master/designs/README.md for details.

Note: Configuration variables are defined in many files under different directories. It will be helpful to explore (https://github.com/The-OpenROAD-Project/OpenLane#flow-configuration) them and determine their priorities over each other.

Exercise:

Adding a design gcd.v (uploaded on canvas) and run through the flow. Please check the timing after synthesis and route.

- Please modify the sdc file and create a case where setup time didn't pass, then use the critical as an example and describe how you can fix the timing for that path. Please upload the full timing path and also your answers for fixing the timing violation.
- Please modify the sdc file and find the maximum frequency that passes the timing check after route. Please upload the screenshot of the layout after place and route, and report area and power.
- Please add 10% of rise and fall time in your sdc for the clock and check the results again. Find the maximum frequency that passes the timing check. Any differences from the first experiment?

- OpenLane comes with a convenient mechanism for running flows https://github.com/The-OpenROAD-Project/OpenLane/tree/master/configuration Please go through them and try to understand what those parameters control.
    1. Run a test where you set the core utilization target to 55%, what did you observe? Is the flow running ok? If not, why? If so, what is the total area? How about timing?
    2. Turn on the placement timing driven and compare against a base case, any changes in timing? Please describe the timing differences.
    3. Try to run synthesis at a different corner (by changing LIB_SYNTH to a use the same lib defined by LIB_SLOWEST. Observe if the maximum frequency and describe what you saw. How about area and power?
    4. Change CTS_TARGET_SKEW to 150ps (instead of 200ps), observe the timing differences, and describe your observation. Can you explain why?
    5. Change the core aspect ratio to 2:1 instead of 1:1, what did you observe, any changes about timing, area and power? Please upload the layout after changing the aspect ratio and your answers to the above question.

3.6 OpenLane File Structure

```
Openlane
├── configuration # scripts configure variables for each stage
├── dependencies
├── designs       # all designs and running results
├── docker_build
├── docs
├── pdks          # Installed pdks
├── regression_results # theoretical results for preinstalled
designs
├── scripts       # Running scrips for each tools and stages
├── conf.py
├── default.cvcrc
├── env.py
├── flow.tcl      # top level script
├── Makefile
├── run_design.py
```

3.7 More Information

Since OpenLane is an open-source project, potential problems and discrepancy exist even in official documents.

- The-OpenROAD-Project/OpenLane start: https://github.com/The-OpenROAD-Project/OpenLane#setting-up-OpenLane
- OpenLane Document https://openlane.readthedocs.io/en/latest/
- Configuration files and variables https://github.com/The-OpenROAD-Project/OpenLane/blob/master/docs/source/PDK_STRUCTURE.md
- Tcl Command Documentation https://www.tcl.tk/man/tcl/TclCmd/contents.html

References:

1. https://inst.eecs.berkeley.edu/~cs250/fa20/labs/lab1/
2. https://github.com/The-OpenROAD-Project/OpenLane

Acknowledgement:

- Zhengping Zhu (UM-SJTU JI)
- Efabless
- OpenLane team

Deliverables:

- Group Deliverables (Compile everything as a single pdf report besides the code):
  - Based on **part 2**, please submit two tcl codes and the screenshot that show they work with a random test case (at least 8 numbers)
  - Based on **part 3**, please submit
    - Based on 3.4, submit the screenshot of the layout of the test case.
    - Based on 3.5, answer all questions in the Exercise section.
  - Answer the following questions (based on your understanding, feel free to use the internet):
    - How many corners are in the Skywater 130nm technology libraries?
    - In openLane, what layers are power and ground using for the power grid?
    - Is the placement and route time better or worse than synthesis timing? Why is that?
    - Why do we need to use a special buffer for clock tree? example: in openLane, they used `sky130_fd_sc_hd__clkbuf_4`.
    - Describe how the openLane project is constructed into a flow based on your observations and understanding.
- Individual Deliverables
  - Complete the peer-evaluation form appended in the end of this document

Grading Policy

Factor	Percentage
Part 2	10%
Part 3	65%
Questions	25%

ECE4810J System-on-Chip (SoC) Design

Fall 2021

Lab #6 Peer Evaluation Form

Each team member is required to provide a peer evaluation for the team effort of the lab. The score of the peer evaluation should be integers ranging between 0 to 5, inclusively, with 5 indicating the biggest contribution. A score should be given to each team member including yourself according the team member's contribution based on your observation. A brief description of specific contribution of each team member should also be provided. Note this form needs to be finished without discussing with your teammate, if all forms are found the same, then you have to justify how each member makes exactly the same contribution (through demonstration to the instructor).

Name	Level of contributions (0 – 5)	Description of contributions
(yourself)		
Team member 1		
Team member 2		
Team member 3		

Your lab grade is calculated based on the following:
Individual_Average = (sum of all team member's marks) / (size of the student team)
Group_Average = sum of Individual_Average of all team members / size of student team
Individual_Difference = Individual_Average / Group_Average – 1.0
Using the calculated Individual_Difference, we find a factor from the following lookup table.

Individual_Difference	Factor	Individual_Difference	Factor
>=0 and < +10%	1.0	> -10% and < 0	1.0
>= +10% and < +20%	1.1	> -20% and <= -10%	0.9
>= +20% and < +30%	1.2	> -30% and <= -20%	0.8
>= +30%	1.3	<= -30%	0.7

Your final lab grade is calculated by:
Final grade of lab = points for team effort * factor

Note that the final grade of the lab will not exceed the maximum points assigned to the project. If the peer-evaluation form is not submitted, then your individual_difference will be based on the available data (without yours) only, and your final score will be:

Final grade of lab = points for team effort * factor * 95%