## Introduction to Physical Design: OpenLane Colab Notebook

*In this section, you will follow along with the Physical Design flow to generate physical IC layouts from RTL using OpenLane.*

*OpenLane is an open-source toolchain for physical design. At SiliconJackets, we use the Cadence suite, not OpenLane. However, following along with this flow is a good way to get an idea of what the Physical Design team does.*

1. Follow [this link](https://colab.research.google.com/github/efabless/openlane2/blob/main/notebook.ipynb) to create a copy of the OpenLane Colab Notebook.

For in-depth information about the OpenLane flow, refer to the [OpenLane reference manual](https://openlane.readthedocs.io/en/latest/reference/index.html).

2. Run all the steps in the OpenLane Colab Notebook. Make sure to read each part thoroughly so you understand what the Notebook is actually doing.

3. Take screenshots of your Notebook after completing each of the following steps and paste them into the answer box below.

* Floorplan
* PDN Generation
* GDS Streamout

|  |
| --- |
| Floorplan    PDN Generation    GDS Streamout |

4. Describe in your own words what the following flow steps do.

a. Floorplanning

|  |
| --- |
| Floorplanning creates a template for the final chip. It determines the size of the chip by calculating the dimensions, and also creates a “cell placement grid” which determines which maps out where cells must be placed on the chip itself (where each cell in the placement grid is called a site). |

b. Clock Tree Synthesis

|  |
| --- |
| Clock tree synthesis is the process of creating a clock tree, which is a hierarchical set of buffers (things placed on a chip) for a clock signal. This is used to minimize a “clock skew” which is a delay of the clock cycle from a register to another register. These skews can cause errors in the chip, which is usually caused by wire length, clock load (number of gates connected to the same buffer), etc. The CTS creates the cells/buffers and places them in between the gaps to lower the potential clock skews. |

c. Global Route

|  |
| --- |
| Global route creates a plan for the wires between two gates (I/O pins, PDN, etc). This routing of wires occurs throughout the pins, and the result is a “routing guide” of all the wires. These routes are stored in the internal data structures of the EDA tools, so they don’t show up on the image of the proposed chip. |

d. Signoff Static Timing Analysis (Post-PnR STA)

|  |
| --- |
| Static timing analysis ensures that a chip can run at its rated clock speed. It makes sure that it meets constraints on clock and data timings. The “static” in STA means that the chip is not executing any functions. STA is a step where the chip is mathematically analyzed purely based on looking at the physical design and calculating the delay to make sure it meets said constraints and runs at its rated clock speed. |

Clocking a Sequential Circuit

*In this section, you’ll get an introduction to critical paths, setup times, and identifying viable clock frequencies.*

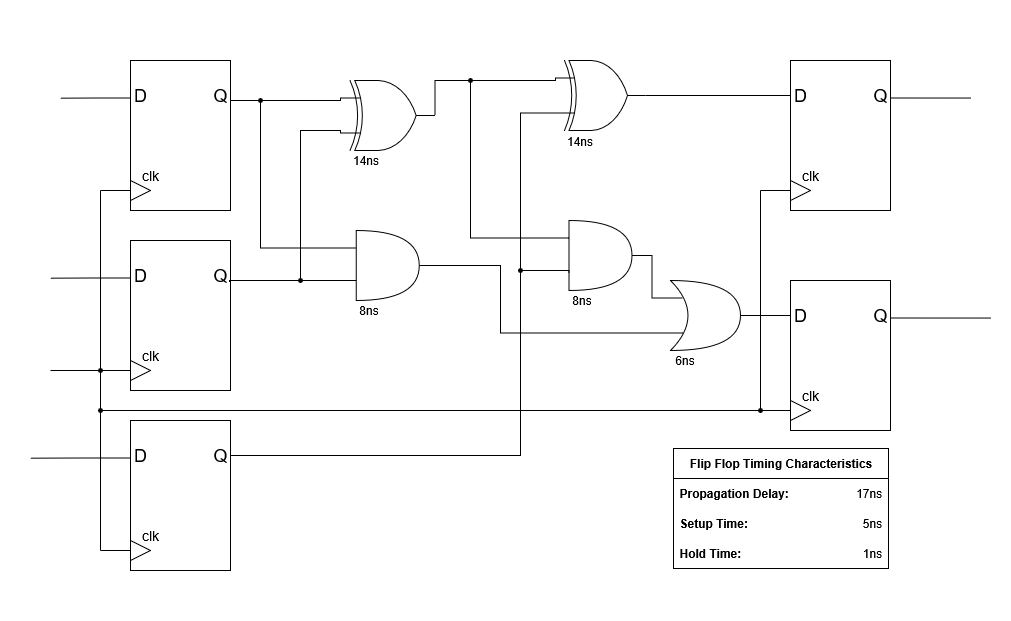
*Performing timing analysis on an IC with millions of transistors requires many of the same essential skills as performing timing analysis on a breadboard circuit diagram — just on a much different scale.*

*This section requires knowledge of some Static Timing Analysis concepts you may not be familiar with.*

*To learn about Setup and Hold times, see* [*this resource*](https://nandland.com/lesson-12-setup-and-hold-time/)*.*

*To learn about Static Timing Analysis, see* [*this resource*](https://anysilicon.com/the-ultimate-guide-to-static-timing-analysis-sta/) *or* [*this more comprehensive resource.*](https://www.eng.biu.ac.il/temanad/files/2018/12/Lecture-5-STA.pdf)

1. For this question, consider the following Sequential Circuit.



How long does it take for a signal to propagate along the circuit’s critical path?

|  |
| --- |
| It takes 50 ns along the circuit’s critical path. |

What is the maximum viable clock frequency?

|  |
| --- |
| 20 MHz |

2. Explain the concepts of setup slack and hold time. Why is a negative slack value problematic?

|  |
| --- |
| Setup slack is the difference between required time (this the time the signal MUST arrive to the flip-flop input at before setup) and arrival time (amount of time a signal actually takes to travel along the circuit). It is a time buffer that allows for the signal to reach the flip-flop and be stable if any sort of interruptions happen along the signal path. Hold time is the amount of time required for an input to be stable in a flip-flop after a clock edge. A negative slack value is problematic because then there is a timing violation, where the arrival time is greater than the required time, meaning the signal takes longer to travel along the circuit, meaning it arrives past the required time which doesn’t allow the signal to be stable before the clock-edge (no setup time). This causes errors in a circuit/chip. |

3. Assume a timing path has a data arrival time of 5ns and a required time of 4.5ns. Is this path meeting timing? Calculate the setup slack. What if the required arrival time was 5.5ns?

|  |
| --- |
| Yes, the path meets timing. The setup slack is 0.5 ns, allowing for 0.5ns of buffer so that the signal arrives at the flip-flop block and becomes stable. If the required arrival time was 5.5 ns, then the path does not meet timing and has a timing violation since the setup slack is now -0.5 ns, meaning the signal will not be stable as there is less setup time than allotted for. |

## Python Timing Report Analysis

*In this section, you’ll get an introduction to the analysis of timing reports generated by our EDA tools using Python.*

*We’ve provided a sample timing report (full\_paths.max.rpt) for you to analyze.*

*The report is too large to analyze by hand, so you’ll have to write a Python script to help parse the file data. You can use the provided template.py if you’d like.*

[*This guide on File I/O*](https://www.tutorialspoint.com/python/python_files_io.htm) *and* [*this guide on String processing*](https://www.pythoncentral.io/how-to-parse-a-string-in-python-a-step-by-step-guide/) *might be good places to start if you don’t have any Python experience.*

1. How many paths violated timing constraints (i.e., have negative slack)?

|  |
| --- |
| 50 paths have negative slack/violated timing constraints |

2. What is the average depth (or number of gates) of paths with the smallest positive slack?

**Hint:** In this context, “smallest positive slack” refers to paths that *are* passing timing, but just barely.

Answering this question might involve creating a script that extracts tuples containing each path’s slack and gate depth, then sorting the tuples by slack.

|  |
| --- |
| 8 paths with least positive slack(0), and the average depth was 17 gates.  39 paths with least positive slack(0.01), and the average depth was 13.85 |

3. What is this design's operating frequency? From a timing perspective, is there potential to achieve higher performance?

|  |
| --- |
| The operating frequency of the design is 1GHz. There is not a potential to achieve higher performance as there are a lot of timing violations, meaning that the design must be fixed before even considering increasing the frequency (also we need to increase the period to take care of the timing violations, which lowers the frequency, so we can’t increase the operating frequency). |

**Advanced Questions** *- please try to answer as much as you know without the use of the internet*

4. This timing report comes from the signoff stage of design (after routing). What is the first issue you notice in this report?

|  |
| --- |
| It’s the use of an ideal clock network. An ideal clock network assumes that the clock signal reaches every single flip-flop in the design at the exact same time. This is a simplification that is used in early stage analysis. After routing, which is after using a clock tree (buffers and wires), this ideal clock network becomes completely obsolete since it ignores the potential clock latency and skew.  The analysis must use a propagated clock to model the real delays and skew for the clock tree. |

5. How would you address the issue identified in question 4? Additionally, reconsider your answer to question 3 based on this observation.

|  |
| --- |
| To fix the issue in question 4, we need to change the settings in our timing analysis tool to the use the clock-tree rather than an ideal clock network. This means that increasing the clock speed of the chip becomes a lot more challenging and there is even less potential to increase the performance as by using the clock-tree we would find a lot more timing violations (negative slack). |

TCL Procedures

*This section focuses on the Tool Command Language (TCL), which we use to write scripts that automate aspects of running the flow.*

*You need to run the flow to completion before answering some problems in this section. See* [*the RTL2GDS README in the repo*](https://github.gatech.edu/SiliconJackets/Physical-Design-Onboarding/tree/main/RTL2GDS#rtl2gds-tutorial) *for instructions on running the flow.*

1. Navigate to the *design/scripts* directory. Choose a *.tcl* script in this directory and explain what it does.

|  |
| --- |
| *add\_pins.tcl* is a file that marks and edits the pin placement. It can fix pins (position will not change during optimization), it can apply a change to all pins in a design, it can specify some placement pattern for the pins in the design to optimize them for filling space, and can add the pins and assign them to specific metal layers for routing purposes. |

2. We want to highlight an SRAM block in our design.

a. Consult the Cadence Support website to identify the Innovus commands to dim the background and highlight an object.

Use these commands in the Innovus shell to highlight an SRAM block and paste a screenshot in the box below.

|  |
| --- |
|  |

b. Let's automate that process. Create a TCL proc (*procedure*) that takes an SRAM’s name as an input and invokes these two commands to highlight it in Innovus.

Paste or screenshot your TCL proc code here, as well as the file being sourced in the Innovus shell.

|  |
| --- |
| proc highlight\_sram { sram\_name } {    gui\_highlight $sram\_name    gui\_dim\_foreground -light\_level dark  } |

2. We can also use TCL to highlight paths in our design. This is especially useful for highlighting paths that fail timing to help us decide how to optimize our layout to improve timing.

Find and use Innovus commands that read a timing debug report and highlight the paths in the GUI. Paste a screenshot of the GUI showing failed paths in the box below.

|  |
| --- |
|  |

3. Let’s automate this too. Write a TCL proc that takes in a path number as an input and highlights in the GUI.

Paste or screenshot your TCL proc code here, as well as the file being sourced in the Innovus shell.

|  |
| --- |
| proc highlight\_path { x } {      gui\_gtd\_highlight\_timing\_report -in\_file toplvl.btarpt -path $x  } |

Static Timing Analysis on our 32-bit Adder

*The questions in this section involve analyzing the timing reports generated by running the flowtool in Github repository on the 32-bit adder design.*

*You should have already run the flow once, and you will have to run it a few more times. In case you haven’t run it yet, see* [*the RTL2GDS README in the repo*](https://github.gatech.edu/SiliconJackets/Physical-Design-Onboarding/tree/main/RTL2GDS#rtl2gds-tutorial) *for instructions on running the flow.*

4. Identify the Worst Negative Slack (WNS) from the timing summary for both setup and hold in the current design.

**Hint:** *There are multiple ways to find this information. The file qor.html may be a good place to start.*

|  |
| --- |
| Setup WNS: -0.206 ns  Hold WNS: -0.206 ns |

5. Using the WNS and the clock frequency (set in the file *pipeline\_src/base.sdc*), calculate the max frequency that the current design can theoretically run at.

**Hint:** *Frequency is the reciprocal of period.*

|  |
| --- |
| Clock Speed = 1/(1-(-0.206))ns 829.19 MHz |

6. Modify the current design SDC file and rerun the flow to make the WNS at least +0.01ns. Include the screenshot of the flow summary or *qor.html* showing your positive WNS.

|  |
| --- |
|  |

Placing Macros in Innovus

*At this point, our design is meeting timing!*

*Does that mean our chip is ready for tape-out? No, unfortunately not.*

*This section will familiarize you with Design Rule Checking (DRC), another challenge we have to consider in the Physical Design process.*

*In this section, you will be implementing and understanding the importance of macros in VLSI design. Within your Verilog design, you instantiate SRAMs to use as memory blocks. These cannot be implemented with RTL; instead, we have to instantiate them as design macros.*

*For help using Innovus, see the* [*Using Innovus guide*](https://github.gatech.edu/SiliconJackets/Physical-Design-Onboarding/blob/main/docs/Using-Innovus.md#using-cadence-innovus) *in the docs/ directory.*

1. If you haven’t already, do a complete run-through of your flow and open the final (postroute) database in Innovus.

The white markers in the design are what are known as DRC violations. DRC stands for Design Rule Checking.

Foundries supply chip designers with design rules that specify requirements the design must meet for it to be taped out. Design libraries often have hundreds of rules. Some examples are minimum spacing between elements, layer overlap, and area limitations.

These are rules the foundry supplies us with allowing us to know whether a chip can actually be manufactured. Your goal as a PD engineer should be to create a manufacturable design, and in doing so you have to correct the DRC violations you've found. Describe what you see and make some inferences about what has caused them. Additionally, provide an image of the design.

Provide a screenshot of your design’s DRC violations in Innovus, describe what they look like, and make some inferences about what has caused them.

|  |
| --- |
| Violation Types:   * *Antenna Area Ratio & Antenna Side Area Ratio (solved later on as it does not show up in Verify errors)* * **Metal Short: This is a connectivity violation. It indicates that two or more different metal traces on the same layer or different layers are electrically connected (shorted) when they should be isolated.** * **Parallel Run Length Spacing: This is a spacing violation. Design rules often specify a minimum spacing between parallel metal traces, and this minimum spacing can increase if the parallel running length exceeds a certain threshold. This rule helps to prevent crosstalk and capacitive coupling between adjacent wires.** |

*The DRC violations can be fixed through modifications to our TCL code.*

*In the previous section, you learned some examples of TCL and what it can be used for. For the rest of this section, you will be trying to modify TCL code to fix some of the errors you encountered.*

2. Using the Cadence Support website, look for a command to be able to manually place these SRAM instances in the design.

In the box below, provide these commands and give a brief description of why you think we are placing these macros manually instead of letting the tool decide.

|  |
| --- |
| *place\_inst*  I think we are using manual commands to place instances in chip design to guide the automated tools and achieve better performance, especially for critical parts of the circuit. While automated place-and-route tools are incredibly powerful, they don't have a complete understanding of the design's intent. Manual placement gives the designer (us) control over a few key aspects that the tools might otherwise get wrong. |

3. This is one of the three distinct commands necessary to fix the DRC errors you have encountered thus far.

As you saw in the OpenLane tutorial, during the physical design flow, the tool **places** cells throughout the design and **routes** physical connections between cells in your design.

However, when your design utilizes macros, this can sometimes cause problems, as it attempts to **place** and **route** too close to the macro.

You will need two more commands to block the cells and metal layers from causing DRCs. Use Cadence Support to find the commands you are going to use and describe how they will prevent errors in your design.

|  |
| --- |
| *create\_place\_blockage*  This command creates a buffer zone around our macro. The automated placer will not place any standard cells in this zone, which directly prevents spacing and short violations between the macro's pins and any nearby cells.  *create\_route\_blockage* This command used to stop the router from creating wires over or too close to the macro, which can cause metal spacing DRCs or noise issues. By blocking the lower and mid-level metal layers over the macro's area, we force signals to route around it, keeping the area clean. |

4. Now that you’ve found the commands necessary to fix the DRC violations, it's time to fix them. To do this you will be taking the commands that you found and implementing them in the TCL code.

In *floorplan.tcl* there are two commented lines labeled with “REQUIRED”. Under each of those, add your implementations for placing your SRAMs and preventing DRC violations. Here is some helpful information for your implementation:

* Each SRAM block is 683nm x 416nm in size
* You must keep in mind that the metal 5 layer will be used to provide power to the SRAM blocks

Once you have your implementation set in the scripts, run the flow again. You can run from the floorplan step as nothing before that step has been modified.

Attach a picture of your final design without any DRC violations!

|  |
| --- |
|  |