# Exploring Parallelism Opportunities in KLayout (project proposal, group 23)

Zeren Chen*
TBD@TBD.tw
Taiwan

MH Yan
TBD@TBD.tw
Taiwan

ChatGpt
Open AI
USA

## KEYWORDS

KLayout, DRC, Parallel Programming, Open-Source

## 1 INTRODUCTION

Design Rule Checking (DRC) is a crucial step in the VLSI design process, ensuring that the layout adheres to the specified design rules. Some basic and common types of DRC rules are:

- Minimum width
- Minimum spacing
- Minimum area

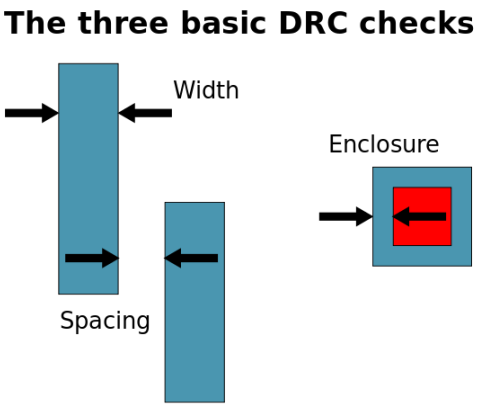


**Figure 1: The three basic DRC checks (source: Wikipedia)**

KLayout(https://www.klayout.de), an open-source layout viewer and editor, is widely used for VLSI design, including DRC feature support. However, as the complexity of IC designs increases, the computational demands of KLayout's DRC also increase. This project aims to learn how DRC works in Klayout, explore opportunities for parallelism within KLayout using threading, SIMD techniques, and advanced parallel programming techniques such as OpenMP, MPI, and CUDA, if time permits. Additionally, we will analyze the DRC run set written in Python to find the possibility of task parallelism. Furthermore, we will attempt to reach out to the KLayout author to gather valuable feedback and insights to guide our project.

## 2 STATEMENT OF THE PROBLEM

The primary focus of this project is to identify areas within KLayout's DRC where parallelism can be applied to improve its performance, particularly in terms of processing large and complex layout designs. To achieve this, we will analyze the existing KLayout codebase, focusing on the DRC run set written in Python/Ruby, and identify bottlenecks and opportunities for parallelization, including task parallelism, in the DRC process.

**Listing 1: Sample DRC code in KLayout**

```
report("A simple script")

active = input(1, 0)
poly = input(6, 0)
gate = active & poly
gate.width(0.3).output("gate_width", "Gate width violations")

metal1 = input(17, 0)
metal1.width(0.7).output("m1_width", "M1 width violations")
metal1.space(0.4).output("m1_space", "M1 spacing violations")
```

This script will compute the gate poly shapes from the active and poly layers using a boolean "AND". The active layer has GDS layer 1, while the poly layer has GDS layer 6. On this computed gate layer, the script will perform a width check for 0.3 micrometer. In addition a width and space check is performed on the first metal layer, which is on GDS layer 17.

## 3 PROPOSED APPROACHES

We will first conduct a detailed code analysis of KLayout's DRC, focusing on the Python/Ruby run set, to identify computationally intensive tasks and potential areas for parallelization. For example, C++ code that implements width, space operations of Listing 1. Next, we will design and implement a parallelized version of KLayout's DRC using threading techniques (e.g., Pthreads, OpenMP) and SIMD (Single Instruction, Multiple Data) instructions. If time permits, we will also explore the possibility of employing advanced parallel programming techniques such as MPI and CUDA learned in the class. We will then evaluate the performance of the parallelized DRC by measuring the time taken to process various layout designs generated by scripts written by us or some real open source CMOS designs available.

## 4 LANGUAGE SELECTION

Ideally, we would like to apply all the techniques learned in the class to improve the run time of certain DRC C++ code called by DRC runset written in Python/Ruby.

- SIMD
- Threading
- OpenMP

*Still trying to reach out to teammate

- MPI(less likely as klayout only works on single host)
- CUDA

We will also explore the possibility of using the Taskflow library, a modern C++ parallel and heterogeneous task programming library, to achieve parallelism in KLayout's DRC.

## 5 RELATED WORK

A preliminary literature search indicates limited research on parallelization opportunities within KLayout's DRC. As a result, our project will provide valuable insights into improving the performance of KLayout's DRC and contribute to the growing body of knowledge in the field.

## 6 STATEMENT OF EXPECTED RESULTS

Our primary focus is on learning and gaining experience working with large codebases and open-source communities. We expect to identify potential areas for parallelization within KLayout's DRC, including task parallelism, and attempt to implement a parallelized version using threading, SIMD techniques, and advanced parallel programming techniques (if time permits). Although our work may not be accepted upstream, we aim to demonstrate improved performance in terms of processing time for large and complex layout designs, and to acquire valuable insights and practical knowledge through our exploration of KLayout's DRC.

## 7 TENTATIVE TIMETABLE

- April 7 - May 31, 2023:
  - Reviewing the source code of KLayout and find portions that can be parallelized.
  - Prepare test run set, test cases, benchmarking serial versions.
  - Work on threading, SIMD, openMP, CUDA, taskflow versions respectively.
- June 1 - 15, 2023: Prepare final report and presentation slides
- June 16, 2023: Submit final report, source codes

## 8 REFERENCES


- KLayout Documentation, "KLayout User Manual," [Online]. Available: https://www.klayout.de/doc/index.html [Accessed: 04-April-2023].
- KLayout Repository, "KLayout Source Code," [Online]. Available: https://github.com/KLayout/klayout [Accessed: 04-April-2023].
- Taskflow, "Taskflow: A General-purpose Parallel and Heterogeneous Task Programming System," [Online]. Available: https://taskflow.github.io/ [Accessed: 04-April-2023].