

Exploring Parallelism Opportunities in KLayout

A Journey of Learning with Open Source EDA Software

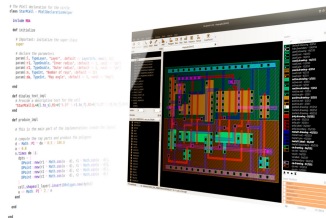
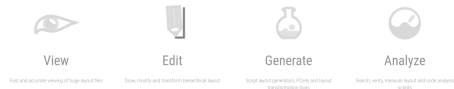
Group 22: 陳則仁 & 鄢銘宏

June 16, 2023

• About KLayout:

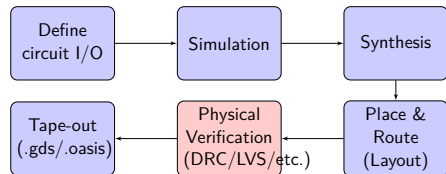
- A open source (GPL 3.0) project started as a viewer, then editor/analyzer/multitool
- First released in 2006-04; By Matthias Koefferlein
- Cross platform (Linux distros/Windows/MacOSX) achieved by QT, compact <100MB installation
- Users from Google open source PDK project, special manufacturing process like photonic, quantum computer. Also widely used by personnels in the EDA (Electrical Design Automation) industry.
- Commercial alternatives include Virtuoso(CDNS) or Laker(Was Spring-Soft, later acquired by SNPS) for layout/schematic editing/viewing, Calibre(Siemens EDA) for physical verification

KLayout - Your Mask Layout Friend



Project Goals

- Apply parallel programming techniques to improve runtime of KLayout's Design Rule Check (DRC) operation
- Gain insights from a substantial, real-world C++ code base
(`git ls-files |grep ".cc"| xargs wc -l` yields 2,688,261 lines of code)
- Explore and learn from the open-source EDA ecosystem



Inputs: Layers, Polygons, and Cells

- **Layer:** Layers in a GDSII or OASIS layout input can be visualized as a stack of transparent films, each containing different geometries.
- **Polygon:** The fundamental building blocks of the layout are polygons, basic shapes that are used to form more complex structures, it consists of points at 2D plane.
- **Cell:** A cell can contain multiple layers with different polygons, forming a hierarchical structure in the layout design.

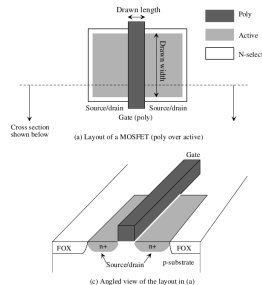


Figure 4.3 Layout and cross-sectional views of a MOSFET.

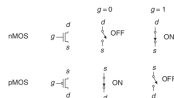


Figure 1.31 Switch models of MOSFETs



Figure 1.32 NOT gate schematic

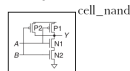


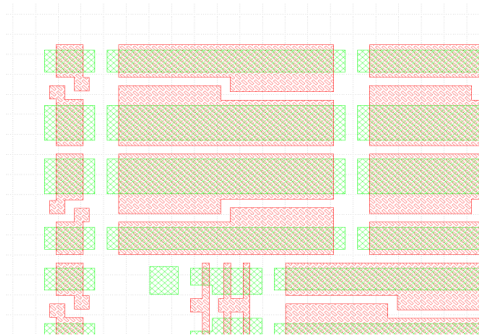
Figure 1.33 Two-input NAND gate schematic

Example DRC Rules (Skywater 130)

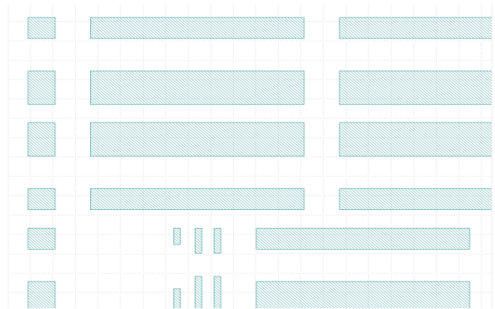
- `poly.not(diff).edges.and(gate.and(lvttn).edges).space(0.35, euclidian).output("poly.1b", "poly.1b: min. lvttn gate width : 0.35um")`
- `poly.isolated(0.21, euclidian).output("poly.2", "poly.2 : min. poly spacing : 0.21um")`
- `poly.and(rpm.or(urpm).or(poly_rs)).width(0.33, euclidian).output("poly.3", "poly.3 : min. poly resistor width : 0.33um")`

These rules can serve as examples to understand the layout of DRC rules. DRC rules are written in a domain-specific language, which allows expressing complex geometric constraints in a compact and readable way. In these examples, geometric operations are combined with logical operations to describe the required conditions on layout layers.

Results of "and" operation



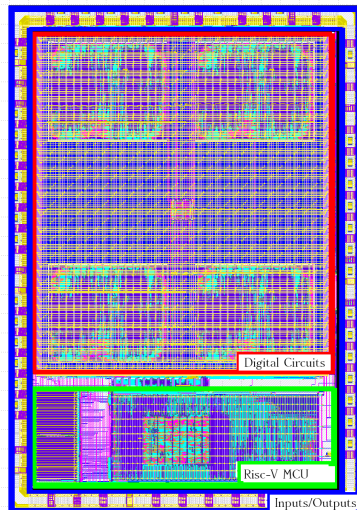
green: diffusion(diff)
red: poly layer



poly and diff

Test case from SkyWater 130 Process and Open Source IC Design

- **SkyWater 130 Process:** An open-source semiconductor process for designing custom ICs. SkyWater 130nm provides a PDK for creating manufacturable designs.
- **Google Open Source IC design project:** Google and eFabless have partnered to produce open source designs that can be manufactured using the SkyWater 130 process.
- **Test Case:** Many Caravel RISC-V test cases can be found at [github](https://github.com).



Proposed Solutions

- **OpenMP:**

- *Benefits:*

- Easy to integrate into existing code.

- *Drawbacks:*

- Requires careful management of shared data to avoid race conditions.
 - Not successful in this particular scenario.

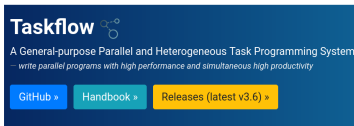
- **TaskFlow:** Task, DRC operation level parallelism.

- *Benefits:*

- Provides more fine-grained control over parallelism compared to OpenMP.
 - Can express complex dependencies between tasks.

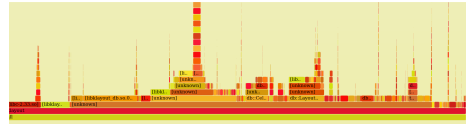
- *Drawbacks:*

- More complex to set up and manage compared to OpenMP.
 - Work-in-progress for this particular scenario.



Lessons Learned from OpenMP and Klayout DRC

- Objectives: Attempted to improve the runtime of “gate = poly.and(diff)” operation of caravel test case. ‘(5 Million gate shapes)’
- Challenges faced:
 - Code is hard to trace. RBI calls Cpp code with templates, overrides, delegate patterns. One segment of code is shared by many DRC commands (and, or implementations are shared).
 - Used gdb/perf+flameGraph to understand where the slowest part is.
 - Iterator pattern is widely used, making it difficult to apply parallel for pragma. Once found a for loop in the scanline algorithm in and_or operation that seemed to be a good parallel for candidate but later turned out that part of code is not reentrant/thread safe.



Flamegraph when running with klayout DRC

Work in Progress with TaskFlow + Klayout DRC

- TaskFlow allows for easy task management by simply declaring them, while effectively handling parallelism.
- Given that DRC rules are largely independent and can be modeled as distinct tasks, TaskFlow offers a promising approach to enhance runtime without requiring extensive modification to the Klayout code base.

```
#include <taskflow/taskflow.hpp> // Taskflow is header-only

int main(){

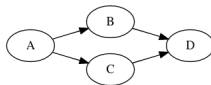
    tf::Executor executor;
    tf::Taskflow taskflow;

    auto [A, B, C, D] = taskflow.emplace( // create four tasks
        [] () { std::cout << "TaskA\n"; },
        [] () { std::cout << "TaskB\n"; },
        [] () { std::cout << "TaskC\n"; },
        [] () { std::cout << "TaskD\n"; }
    );

    A.precede(B, C); // A runs before B and C
    D.succeed(B, C); // D runs after B and C

    executor.run(taskflow).wait();

    return 0;
}
```



KLayout Code with TaskFlow

- This is an example of KLayout's C++ code that utilizes TaskFlow.
- Compared to the sequential version, there is approximately a 4X improvement in runtime.

```
db::Layout ly;
{
    std::string fn (tl::testdata ());
    fn += "/algo/deep_region_l1.gds";
    tl::InputStream stream ("/home/zeren/ws/klayout_workspace/klayout/decired_controller.gds.gz");
    db::Reader reader (stream);
    reader.read (ly);
}
tf::Executor executor;
tf::Taskflow taskflow("simple");

db::cell_index_type top_cell_index = *ly.begin_top_down ();
db::Cell &top_cell = ly.cell (top_cell_index);
unsigned int lm1 = ly.get_layer (db::LayerProperties (68, 20));
unsigned int lv1a = ly.get_layer (db::LayerProperties (68, 44));
unsigned int lm2 = ly.get_layer (db::LayerProperties (69, 20));
unsigned int ldifff= ly.get_layer (db::LayerProperties (65, 20));
unsigned int lpoly= ly.get_layer (db::LayerProperties (66, 20));

db::Region all_diff (db::RecursiveShapeIterator (ly, ly.cell (top_cell_index), ldifff));
db::Region all_poly (db::RecursiveShapeIterator (ly, ly.cell (top_cell_index), lpoly));
db::Region all_m1 (db::RecursiveShapeIterator (ly, ly.cell (top_cell_index), lm1));
db::Region all_via (db::RecursiveShapeIterator (ly, ly.cell (top_cell_index), lv1a));
db::Region m1_and_via, l1, l2, l3, l4, gate;

auto [A, B, C, D] = taskflow.emplace(
    [&l1, &all_m1, &all_poly]() { l1 = all_m1 & all_poly; std::cout << "TaskA\n"; },
    [&l2, &all_m1, &all_diff]() { l2 = all_m1 & all_diff; std::cout << "TaskB\n"; },
    [&l3, &all_via, &all_poly]() { l3 = all_poly & all_via; std::cout << "TaskC\n"; },
    [&l4, &all_m1, &all_via]() { l4 = all_m1 & all_via; std::cout << "TaskD\n"; }
);
```

Conclusion and Future Work

- The KLayout DRC operation can leverage parallel programming techniques for enhanced performance.
- While OpenMP provided a good starting point, it proved challenging due to the complexities inherent to the codebase.
- TaskFlow demonstrates potential in achieving task/DRC operation level parallelism.
- Future work encompasses continued implementation of TaskFlow, parsing Ruby code into syntax trees to generate C++ TaskFlow-based KLayout DRC code, and further exploration of the codebase to uncover additional parallelism opportunities.

Thank you for your attention! Questions?