Andreas Krinke
Institute of Electromechanical and Electronic Design, Dresden University of Technology

Layout Verification Using Open-Source Software

International Symposium on Physical Design
Taipei, March 14, 2024

andreas.krinke@tu-dresden.de
www.ifte.de
Outline

1. Motivation
   - Open-Source Software for Layout Verification
   - OpenPDKs
2. Generating DRC/LVS Runsets for KLayout
   - Data Structure
   - KLayout Generator
   - Current Status
3. Conclusion & Outlook
Motivation: Problems with Commercial EDA Tools

- Licensing Costs
  - High licensing costs, higher for smaller technology nodes
  - Discounts, maybe?

- Confidentiality
  - Example: Trustworthy electronics (e.g. for cryptography)
  - For software: Reproducible builds – from source to binary
  - For integrated circuits: mask layout is confidential
Motivation: Layout Verification

- Before production: Verification of mask layouts
  - Design Rule Check \( \text{DRC} \)
  - Layout vs. Schematic \( \text{LVS} \)
  - Parasitic Extraction \( \text{PEX} \)

- Technology information:
  - Part of a foundry's \( \text{PDK} \)
  - Available in proprietary formats (e.g. \( \text{SVRF} \))
  - Again: High licensing costs for required software tools

- Our goal:
  \textit{Lower barriers to entry for smaller companies – How?}
Open-Source Software for Layout Verification

**KLayout:** DRC, LVS

**Magic VLSI:** DRC, LVS, PEX

**PEX:** FastFieldSolvers (FastCap, FastHenry, ...), OpenRCX
OpenPDKs

- Problem: Open-source tools require OpenPDKs

- OpenPDK: PDK that supports open-source tools; *independent of its license*

- Examples:
  - SkyWater 130 nm
  - GlobalFoundries 180 nm

- Our goal: Generating DRC and LVS runsets for KLayout

Why?
Motivation: Why KLayout?

- Author: Matthias Köfferlein (+ contributors)
- First official release in 2006
- > 500 k LOC (C++); estimated cost: $24 M
- KLayout DRC/LVS scripts are written in Ruby
  - Support for many typical DRC operations: antenna checks, density, connectivity, ...
  - Extensible
- Support for parallelization
- Comprehensive documentation
- Strong copyleft license (GPL)

In theory, no limits for what we can achieve
Motivation: Why Generate?

If we would write the KLayout DRC/LVS runset by hand ...

- Large variety of DRC commands → extensive KLayout scripts
- Possibly a lot of code for “simple” checks
- Mitigation:
  - Custom functions and methods
  - Modularization
- Still: Great effort for new technologies
Our Approach

- Reference technology: **X-FAB XH018 180 nm** technology
- Input format:  KLayout script (Ruby) with (1) extended API, and (2) irrelevant command order
- Target format:  KLayout DRC/LVS script (Ruby)
(Internal) Data Structure

DRC Runset

green = blue . and(yellow)

Parsing & Data Transformation

Internal Data Structure

```json
{
  "type": "layer_definition",
  "layer_name": "green",
  "layer_operation": {
    "type": "layer_operation",
    "keyword": "and",
    "layers": [
      "blue",
      "yellow"
    ]
  },
  "source_line": 1
}
```

JSON + JSON Schema
KLayout Generator: Internal Data Model

- Can an object be represented in KLayout?
- Are all required arguments available?
- Is an object part of a design rule?
KLayout Generator: Simple things are simple

Layer assignments

\[
\begin{align*}
\text{name} & \quad = \quad \text{input(number)} \\
\text{blue} & \quad = \quad \text{input(1)}
\end{align*}
\]

Layer definitions

\[
\begin{align*}
\text{name} & \quad = \quad \text{layer operation} \\
\text{green} & \quad = \quad \text{blue}.\text{and(yellow)}
\end{align*}
\]

Design rules

\[
\begin{align*}
(\text{layer operation}).\text{output(\text{rule name}, \text{comment})} \\
(\text{green}.\text{and(red)}).\text{output(“BROWN”, “Is that chocolate?”)}
\end{align*}
\]
KLayout Generator: Complicated things are possible

Types of Two-Layer Separation Errors

- Separation Errors
- Intersecting Edges
- Touch Points

Ruby / KLayout

```ruby
class DRC::DRCLayer
  def ext_separation(other, value, ...)
    separation_errors = ...
    intersecting_edges_errors = ...
    touch_point_errors = ...
    return (separation_errors
             + intersecting_edges_errors
             + touch_point_errors)
  end
end

blue.ext_separation(green, 1.0).output(...)
```
Verification on Test Layouts

- Behavior of layer operations verified on test layouts
- Partial or full support of **33 layer operations** from the design manual

Only intended DRC errors
Current Status

Support for 74% of the 1142 design rules of XH018
Conclusion

- Comprehensive DRC \textit{and} LVS for commercial technologies using KLayout
- Including, e.g.
  - Marker Browser
  - Creation of result databases (RDBs)
- \textbf{Input:} Extended KLayout script (Ruby)
- \textbf{Output:} Vanilla KLayout DRC/LVS script (Ruby)

- Our generator is \textit{not} open source; output can be part of an OpenPDK
Outlook

OpenPDK for IHP’s SG13G2 130 nm Technology

- BiCMOS technology, high bipolar performance with $f_t/f_{\text{max}} = 350/450$ GHz
- Our contribution: KLayout DRC script

EM-DRC

- Goal: Electromigration check based on new stress-based EM models
- Inputs: Currents, interconnect geometries
- Tool: KLayout
Outlook

OpenPDK for IHP’s SG13G2 130 nm Technology

- BiCMOS technology, high bipolar performance with $f_t/f_{max} = 350/450$ GHz
- Our contribution: KLayout DRC script

EM-DRC

- Goal: Electromigration check based on new stress-based EM models
- Inputs: Currents, interconnect geometries
- Tool: KLayout

Thank you!
Current Work: (1) LVS

- **LVS = Layout Versus Schematic**
- **4 major steps:**
  - Device recognition
  - Device parameter calculation
  - Connectivity extraction
  - Netlist comparison

- **Support of new device types requires**
  - New `DeviceExtractor` class
  - New `DeviceParameterCompare` class
Current Work: (1) LVS

- LVS = Layout Versus Schematic
- 4 major steps:
  - Device recognition
  - Device parameter calculation
  - Connectivity extraction
  - Netlist comparison

- Support of new device types requires
  - New DeviceExtractor class
  - New DeviceParameterCompare class
Current Work: (2) Assembly Rule Check

- Advanced packaging requires verification of **assembly rules**
- Sources of assembly rules:
  1. **Assembly technology** for individual dies (e.g. pick and place, micro transfer printing)
  2. **Connection technology** (e.g. wire bonding, bumps, micro bumps)
- Our goals:
  - Formal description of packaging technologies
  - Automatic generation of DRC runsets for KLayout

Current Work: (2) Assembly Rule Check

- **Approach**: Extensible Data formats (JSON + JSON Schema) to describe components, assemblies, assembly steps, as well as placement & connection styles.

```
Assembly
{
  metadata,
  "components": {
    "chip": {
      ...
    },
    "interposer": ...
  },
  "assembly_steps": [
    ...
  ],
}
```

```
Component
{
  "layout": "Chip.gds",
  "top_cell": "Chip",
  "top_layer": "1/0",
  "bot_layer": "10/0"
}
```

```
Assembly Step
{
  "top": "chip",
  "bot": "interposer",
  "connection_style": "bumping",
  "placement_style": "pnp",
  "placement": {
    "rotation": 180,
    "position": [5,5]
  }
}
```

```
Connection Style
{
  "technology": "bumping",
  "material": "NiCr",
  "width": 3,
  "square": true,
  "pitch": 7
}
```

```
Placement Style
{
  "technology": "pnp",
  "separation": 5
}
```
Current Work: (2) Assembly Rule Check

- Python program ARC creates custom KLayout DRC script
Open-Source Software for Layout Verification

<table>
<thead>
<tr>
<th>Tool</th>
<th>DRC</th>
<th>LVS</th>
<th>PEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLayout</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Magic VLSI</td>
<td>✓</td>
<td>✓</td>
<td>✓ (R, C)</td>
</tr>
<tr>
<td>Whiteley Research Xic</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>FastFieldSolvers</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenRCX</td>
<td>✓</td>
<td></td>
<td>✓ (R, C)</td>
</tr>
</tbody>
</table>