

KLayout-PEX Documentation

Martin Köhler

2025-01-23

Table of contents

1	Introduction	2
1.1	Motivation	2
1.2	About KLayout-PEX	2
1.3	Installation	3
1.3.1	Option 1: Using IIC-OSIC-TOOLS Docker Image	3
1.3.2	Option 2: Standalone Installation	3
1.3.3	Useful tools: <code>meshlab</code>	4
2	First Steps	4
2.1	Examples	5
2.1.1	Running the FasterCap engine	5
2.1.2	Running the MAGIC-wrapper engine	7
3	Supporting new PDKs	7
3.1	Technology Definition Files	8
3.2	Customized PEX-“LVS” scripts	8
4	Engines: Technical Notes	9
4.1	FasterCap Engine	9
4.1.1	3D Input Geometries	10
4.1.2	Output Maxwell Capacitance Matrix	11
4.2	MAGIC-wrapper Engine	12
4.2.1	Types of Parasitic Capacitances	12
4.3	2.5D Engine	19

1 Introduction

1.1 Motivation

In Electronic Design and Automation (EDA) for Integrated Circuits (ICs), a schematic presents an abstraction in comparison to the layout that will eventually be taped-out and fabricated by the semiconductor foundry. While in the schematic, a connection between device terminals is seen as an equipotential, the stacked geometries in a specific layout introduce parasitic effects, which can be thought of additional resistors, capacitors (and inductors), not modeled by and missing in the original schematic. To be able to simulate these effects, a parasitic extraction tool (PEX) is used, to extract a netlist from the layout, which represents the original schematic (created from the layout active and passive elements) augmented with the additional parasitic devices.

1.2 About KLayout-PEX

KLayout is an open source VLSI layout viewer and editor.

KLayout-PEX (short **KPEX**) is a PEX tool, well integrated with **KLayout** by using its API.

There are multiple PEX engines supported, currently:

- **FasterCap** integration (field solver engine)
- **MAGIC** integration (wrapper calling `magic`)
- Analytical 2.5D engine (parasitic concepts and formulas of **MAGIC**, implemented using **KLayout** methods)

Tip

KPEX tool source code itself is made publicly available on GitHub ([follow this link](#)) and shared under the GPL-3.0 license.

KPEX documentation source code is made publicly available on GitHub ([follow this link](#)) and shared under the Apache-2.0 license.

Please feel free to create issues and/or submit pull requests on GitHub to fix errors and omissions!

The production of the tool and this document would be impossible without these (and many more) great open-source software products: **KLayout**, **FasterCap**, **MAGIC**, **protobuf**, **Quarto**, **Python**, **ngspice**, **Numpy**, **Scipy**, **Matplotlib**, **Git**, **Docker**, **Ubuntu**, **Linux**...

Caution

Currently, **KPEX** is developed as a Python prototype, using the **KLayout Python API**. This allows for a faster development cycle during the current prototyping phase. Eventually, critical parts will be re-implemented (in C++, and parallelized), to improve

performance. As we're already using the KLayout API (which is pretty similar between Python, Ruby and C++), this will be relatively straight-forward.

1.3 Installation

Generally, KPEX is deployed using PyPi (Python Package Index), install via:

```
pip3 install --upgrade klayout-pex

kpex --version    # check the installed version
kpex --help       # this will help with command line arguments
```

As for the dependencies, there are multiple options available.

1.3.1 Option 1: Using IIC-OSIC-TOOLS Docker Image

We provide a comprehensive, low entry barrier Docker image that comes pre-installed with most relevant open source ASIC tools, as well as the open PDKs. This is a pre-compiled Docker image which allows to do circuit design on a virtual machine on virtually any type of computing equipment (personal PC, Raspberry Pi, cloud server) on various operating systems (Windows, macOS, Linux).

For further information please look at the [Docker Hub page](#) and for detailed instructions at the [IIC-OSIC-TOOLS GitHub page](#).

Linux

In this document, we assume that users have a basic knowledge of Linux and how to operate it using the terminal (shell). If you are not yet familiar with Linux (which is basically a must when doing integrated circuit design as many tools are only available on Linux), then please check out a Linux introductory course or tutorial online, there are many resources available.

A summary of important Linux shell commands is provided in [IIC-JKU Linux Cheat-sheet](#).

1.3.2 Option 2: Standalone Installation

- [KLayout](#) layout tool:
 - is mandatory for all engines (besides the MAGIC-wrapper)
 - [get the latest pre-built package version](#)
 - or [follow the build instructions](#)
- [FasterCap](#) engine:

- optional, required to run the FasterCap engine
- either compile your own version from the [GitHub repository](#)
- or use precompiled versions available at <https://github.com/martinjankoehler/FasterCap/releases>
- **MAGIC-wrapper engine:**
 - optional, required to run the MAGIC-wrapper engine
 - Follow the [installation instructions](#) at the [GitHub repository](#)
- **Skywater sky130A PDK:**
 - optional, for now, KPEX technology specific files are deployed within the `klayout-pex` Python package
 - `pip3 install --upgrade volare` (install PDK package manager)
 - `volare ls-remote` (retrieve available PDK releases
 - * for example `PRE-RELEASE 0c1df35fd535299ea1ef74d1e9e15dedaeb34c32 (2024.12.11)`
 - `volare enable 0c1df35fd535299ea1ef74d1e9e15dedaeb34c32` (install a PDK version)
 - PDK files now have been installed under `$HOME/.volare/sky130A`
- **IHP SG13G2 PDK:**
 - optional, for now, KPEX technology specific files are deployed within the `klayout-pex` Python package
 - `git clone https://github.com/IHP-GmbH/IHP-Open-PDK` (install PDK package manager)

1.3.3 Useful tools: meshlab

For previewing generated 3D geometries, representing the input to **FasterCap**, we recommend installing [MeshLab](#).

MeshLab can open the STL-files generated in the `output` directory under `output/<design>/Geometries/*.stl`.

2 First Steps

- The command line tool `kpex` is used to trigger the parasitic extraction flow from the terminal.
- Get help calling `kpex --help`.

2.1 Examples

Example layouts are included in the `testdata/designs` subdirectory of the KLayout-PEX source code:

```
git clone https://github.com/martinjankoehler/klayout-pex.git

# for sky130A
find testdata/designs/sky130A -name "*.gds.gz"

# for IHP SG13G2
find testdata/designs/ihp_sg13g2 -name "*.gds.gz"
```

2.1.1 Running the FasterCap engine

Preconditions:

- `klayout-pex` was installed, see Section 1.3
- `FasterCap` was installed, see Section 1.3

i Note

Normally, devices with SPICE [Nagel_1975] simulation models (e.g. like MOM-capacitors¹ in the sky130A PDK) are ignored (“blackboxed”) during parasitic extraction.

`kpex` has an option `--blacklist n` to allow extraction of those devices (whiteboxing), which can be useful during development (during the prototype phase, whiteboxing is actually the default setting, so please use `--blacklist y` to explicitly configure blackboxing).

Let’s try the following:

```
kpex --pdk sky130A --blackbox n --gds testdata/designs/sky130A/cap_vpp_04p4x04p6_11m1m2_no
```

i Note

This will report an error that we have not activated one or more engines, and list the available engines:

Argument	Description
<code>--fastercap y</code>	Run <code>kpex</code> /FasterCap engine
<code>--2.5D y</code>	Run <code>kpex</code> /2.5D engine

¹Metal-Oxide-Metal capacitors

```
--magic y      Run MAGIC engine
```

Now, to run the FasterCap engine (might take a couple of minutes):

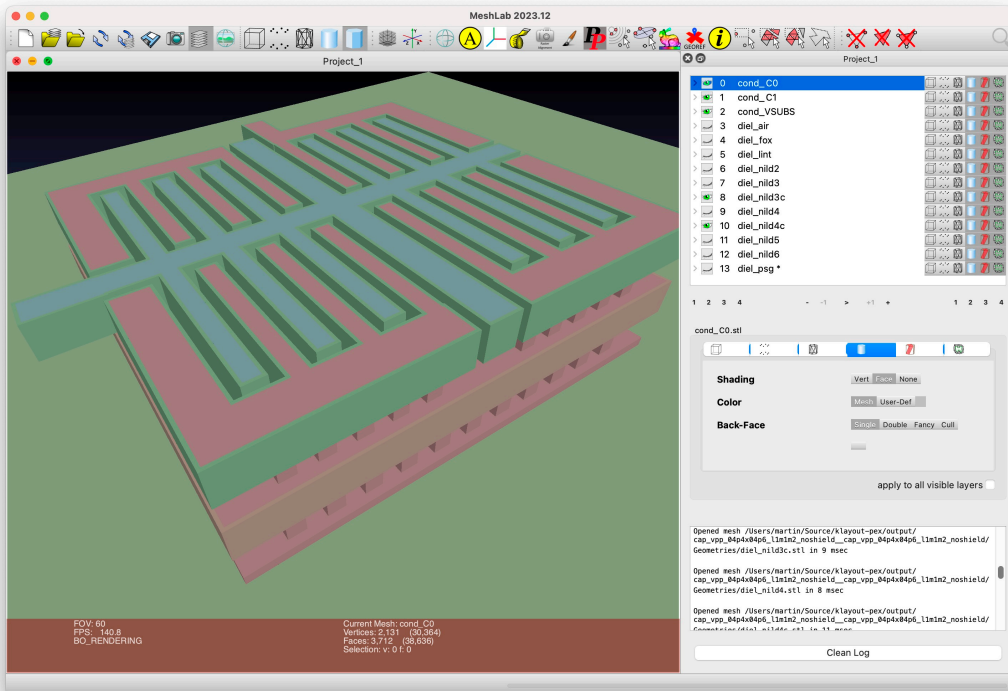
```
kpex --pdk sky130A --blackbox n --fastercap y --gds testdata/designs/sky130A/cap_vpp_04p4x
```

The default output directory is `output`, in there we see a directory `Geometries`, containing STL-files. Those STL files provide a preview of the FasterCap input files. Use MeshLab (see Section 1.3.3) to open and preview those files:

```
ls -d output/cap_vpp_04p4x04p6_11m1m2_noshield__cap_vpp_04p4x04p6_11m1m2_noshield/Geometries
```

Tip

- Open the STL files in MeshLab
- Use the eye buttons to hide and show each file/mesh
- Use the align tool (“A” in the toolbar) to assign different colors
- Start by showing only on the conductors (files named `cond_*.stl`)
- Then try showing different dielectrics (files named `diel_*.stl`), to see how they surround the conductors.



In the log file, we see the output of FasterCap including the Maxwell capacitance matrix:

Capacitance matrix is:

Dimension 3 x 3

```
g1_VSUBS  5.2959e-09 -4.46971e-10 -1.67304e-09
g2_C1    -5.56106e-10 1.5383e-08 -1.47213e-08
g3_C0    -1.69838e-09 -1.48846e-08 1.64502e-08
```

KPEX interprets this matrix and logs a CSV netlist, which can be pasted into a spreadsheet application:

```
Device;Net1;Net2;Capacitance [fF]
Cext_0_1;VSUBS;C1;0.5
Cext_0_2;VSUBS;C0;1.69
Cext_1_2;C1;C0;14.8
Cext_1_1;C1;VSUBS;0.08
```

In addition, a SPICE netlist is generated.

2.1.2 Running the MAGIC-wrapper engine

Preconditions:

- `layout-pex` was installed, see Section [1.3](#)
- `magic` was installed, see Section [1.3](#)

The `magic` section of `kpex --help` describes the arguments and their defaults. Important arguments:

- `--magicrc`: specify location of the `magicrc` file
- `--gds`: path to the GDS input layout
- `--magic y`: enable magic engine

```
kpex --pdk sky130A --magic y --gds testdata/designs/sky130A/cap_vpp_04p4x04p6_11m1m2_noshi
```

3 Supporting new PDKs

For every supported PDK, a KPEX technology definition is required, as well as customized PEX-“LVS” scripts.

3.1 Technology Definition Files

The KPEX technology definition format uses [Google Protocol Buffers](#), so there is:

- formal schema files, defining the structure and data types involved
 - [protos/tech.proto](#): main schema / entry point, includes the others
 - [protos/process_stack.proto](#): describes details of the process stack, such as dielectrics and heights of layers
 - [protos/process_parasitics.proto](#): parasitic tables, used to parametrize the 2.5D engine
- multiple concrete instantiations, that adhere to this schema (called *messages* in the *protobuf* lingo)
 - in the form of JSON files
 - Skywater 130A: `klayout_pex_protobuf/sky130A_tech.pb.json`
 - IHP SG13G2: `klayout_pex_protobuf/ihp_sg13g2_tech.pb.json`

i Note

The built-in JSON tech files are programmatically generated during the build process². Therefore they are not part of the repository source code, but of course part of the deployed Python wheels. To review those, look into your Python `site-packages`³/`klayout_pex_protobuf`.

3.2 Customized PEX-“LVS” scripts

KLayout has built-in support for Layout-Versus-Schematic (LVS) scripts, based on its Ruby API. Customized “LVS” scripts are (“ab”)used in KPEX, not with the intent of comparing Layout-Versus-Schematic, but rather to extract the connectivity/net information for all polygons across multiple layers. The resulting net information is stored in a KLayout LVS Database (“LVSDB”).

²C++ generator scripts the built-in tech files are located in `cxx/gen_tech_pb/pdk/*.cpp`.

³To find the `site-packages` directory for the `klayout-pex` package, call `pip3 show klayout-pex`.

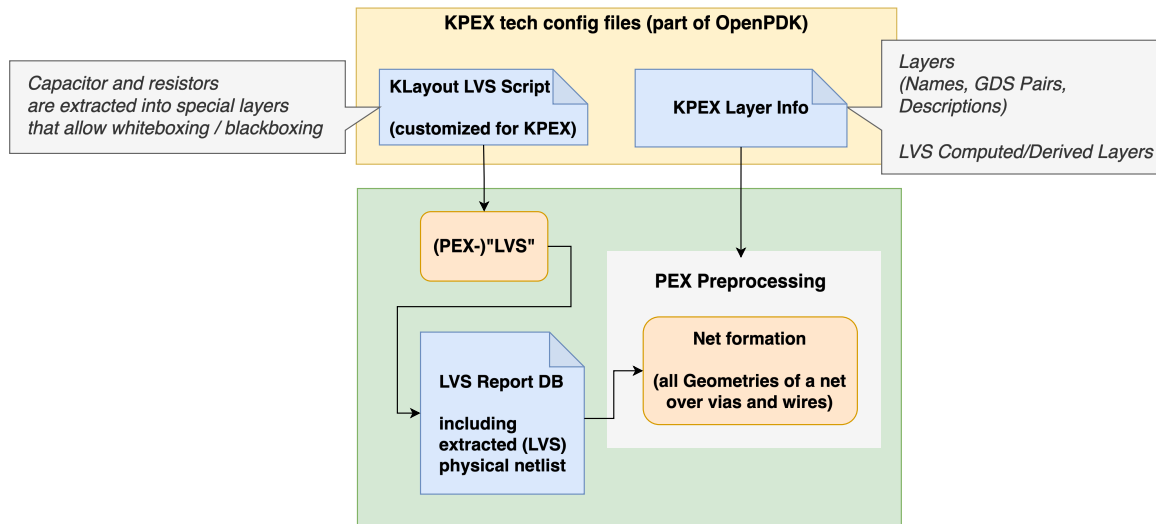


Figure 1: KPEX Net Formation

These customized “LVS” scripts are stored in:

- Skywater sky130A: [pdk/sky130A/libs.tech/kpex/sky130.lvs](#)
- IHP SG13G2: [pdk/ihp_sg13gs/libs.tech/kpex/sg13g2.lvs](#)

What’s specific about this customization:

- Layers names must be assigned, using `KLayout’s (name(layer, name))` function
- MOM⁴ capacitors, MIM⁵ capacitors and resistors should be extracted to separate layers, to enable blackboxing / whiteboxing.

The layer names in the script must correspond with the names configured in the tech JSON file.

4 Engines: Technical Notes

4.1 FasterCap Engine

FasterCap is a 3D and 2D parallel capacitance field solver, inspired by FastCap2. <https://www.fastfieldsolvers.com/fastercap.htm>

Starting from an input layout (e.g. GDS file) and a process stack-up (part of the Section 3.1), KPEX creates input geometries for FasterCap.

After running FasterCap, the Maxwell capacitance matrix is parsed and interpreted to obtain the parasitic capacitances.

⁴Metal-Oxide-Metal capacitors

⁵Metal-Insulator-Metal capacitors

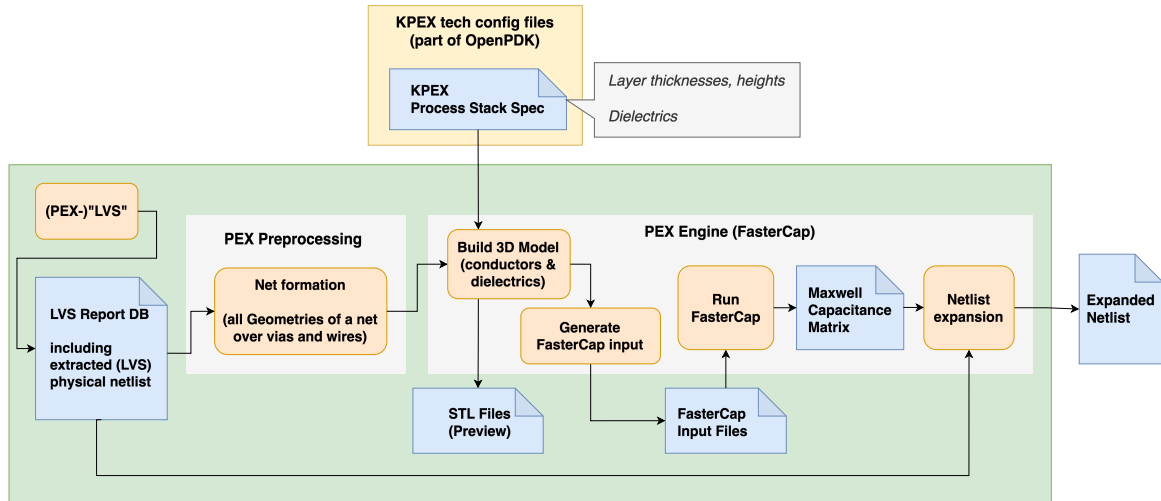


Figure 2: KPEX FasterCap Engine

4.1.1 3D Input Geometries

FasterCap supports 2D and 3D input geometries.



Figure 3: FasterCap 3D Input: File System Overview

KPEX generates 3D input geometries:

- *.lst file: Main input file
 - defines dielectric instances
 - defines conductor instances
 - each instance refers to a *.geo file
- *.geo files: Defines single geometry
 - defines shapes (e.g. triangles)
 - Each shape has a reference point to define inside/outside

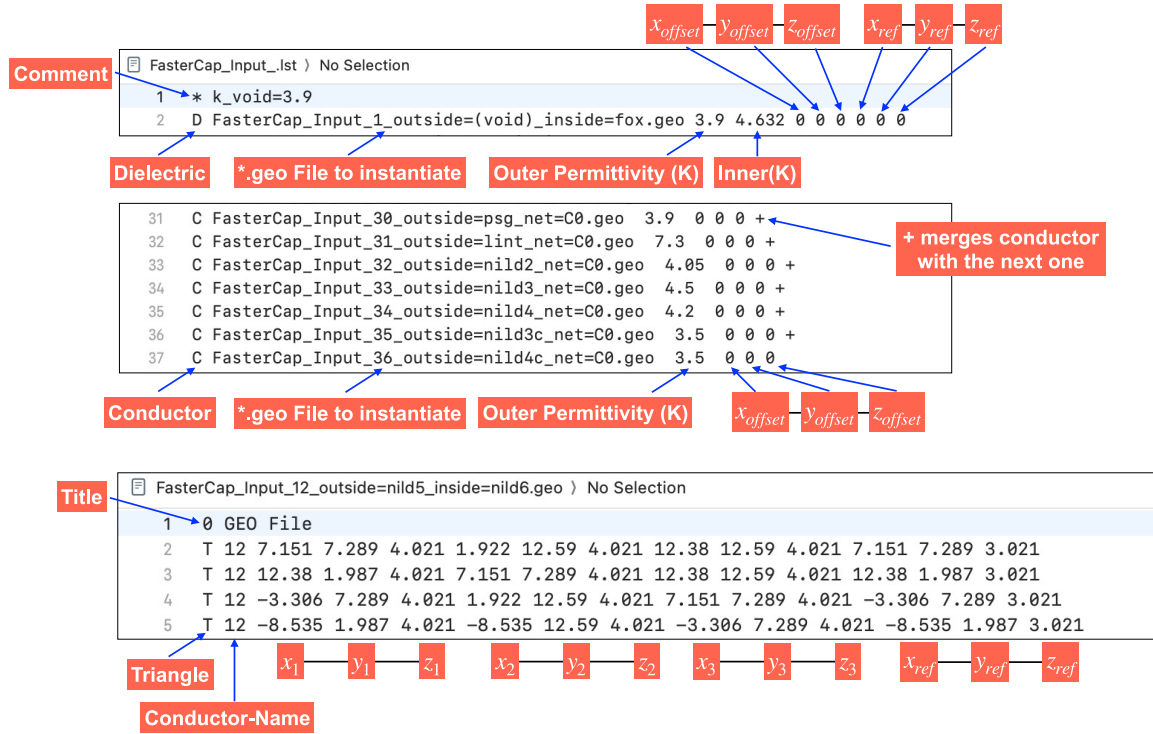


Figure 4: FasterCap 3D Input: File Format

4.1.2 Output Maxwell Capacitance Matrix

A Maxwell capacitance matrix [Maxwell_1873] provides the relation between voltages on a set of conductors and the charges on these conductors.

FasterCap log output prints the Maxwell capacitance matrix (one for each iteration/refinement).

- Matrix Properties:
 - Scaling: units have to be divided by 10^{-6}
 - rows and columns are the same (list of net names)

- Row Cells:
 - * off diagonals cells contains the coupling between row/col nets (times -1)
 - * diagonal cells contains the sum of the absolute values of all other cells in the row
- Matrix Symmetry:
 - * in theory (ideal world), the matrix would be symmetric
 - * in practice it's not
 - * therefore FastCap2 did average the off-diagonals
 - * FasterCap does not average, so it's done as part of KPEX

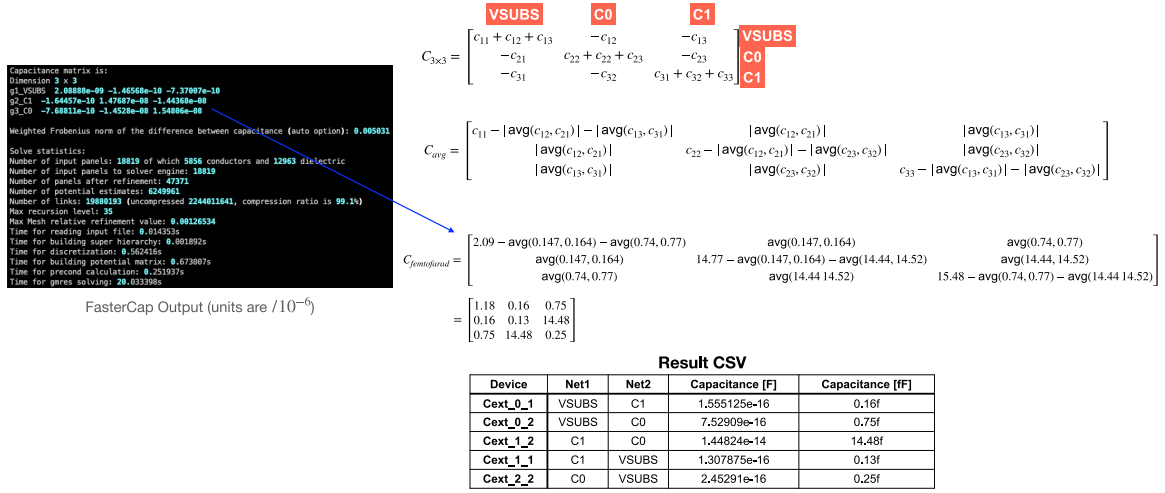


Figure 5: FasterCap Maxwell Capacitance Matrix

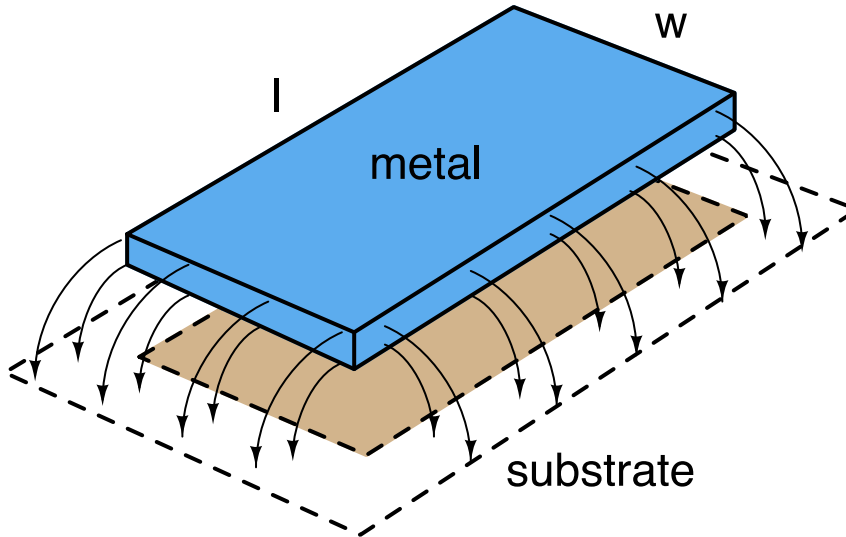
4.2 MAGIC-wrapper Engine

4.2.1 Types of Parasitic Capacitances

MAGIC models multiple types of capacitances:

- *Substrate Overlap*: Overlap area of a metal with the substrate
- *Substrate Fringing*: Sidewall of a metal fringes out to substrate
- *Sidewall Capacitance*: Coupling between adjacent sidewalls on the same layer
- *Overlap Capacitance*: Overlap on different metal layers
- *Fringe Capacitance* (“Side Overlap”): Sidewall of a metal fringes out other metal layers

4.2.1.1 Substrate Capacitance



- Overlapping area:

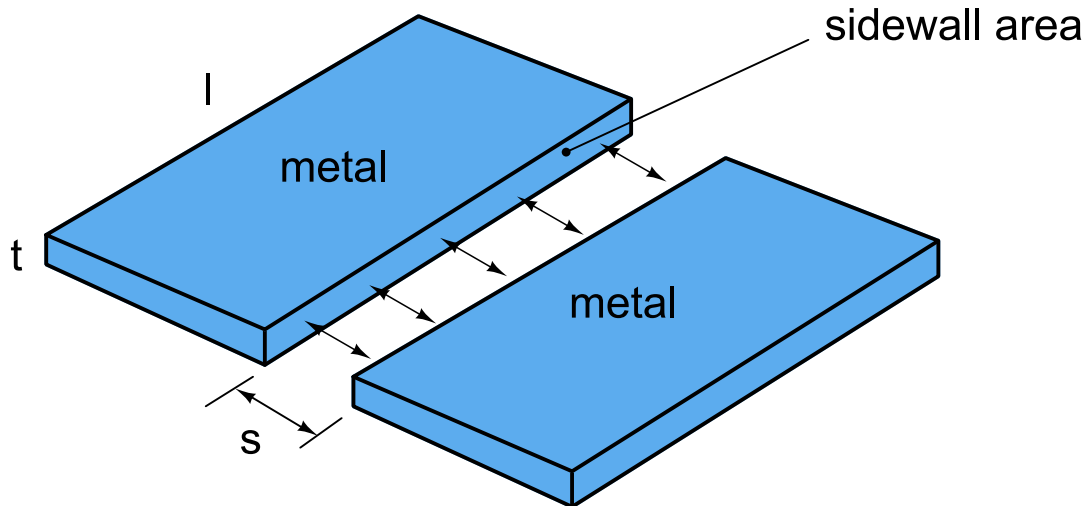
$$C_{area} = \frac{\epsilon_{si} * K}{d} * \text{area} \quad \left[\frac{F}{\mu m^2} * \mu m^2 \right]$$

- Fringe (“Perimeter”):

$$C_{fringe\ to\ substrate} = \text{perimeter} * C_{perim} = (2l + 2w) * C_{perim}$$

Coefficients like C_{perim} are part of the tech files (Parasitic Tables)

4.2.1.2 Sidewall Capacitance



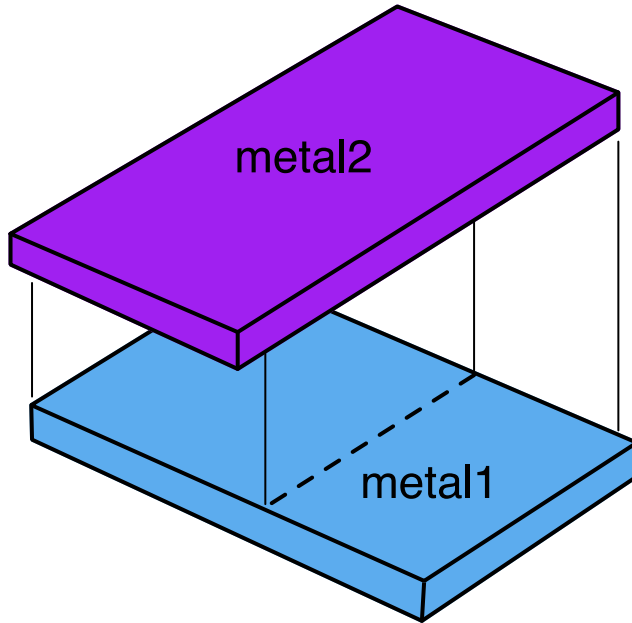
$$C_{sidewall} = \frac{\epsilon_{si} * K}{s} * \text{sidewall area} \quad \left[\frac{F}{\mu m^2} * \mu m^2 \right]$$

$$= \frac{\epsilon_{si} * K}{s} * t * l \quad \left[\frac{F}{\mu m^2} * \mu m * \mu m \right]$$

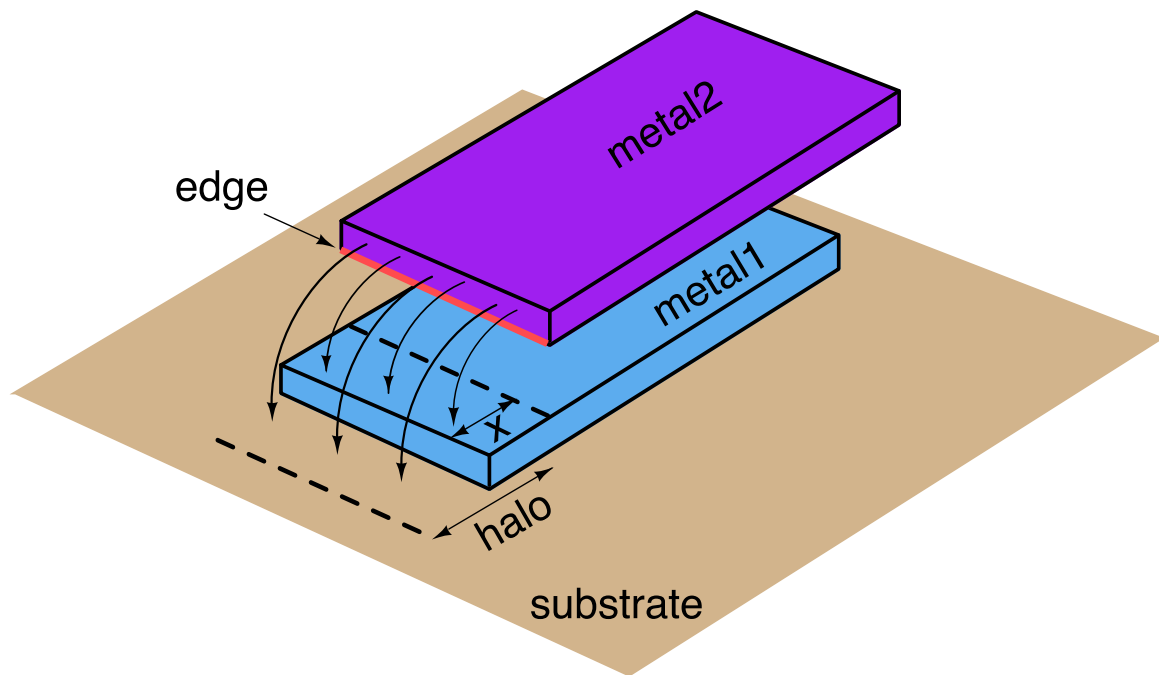
$$C_{sidewall} = \frac{C_{sidewall\ coeff}}{s} * l \quad \left[\frac{F}{\mu m} * \mu m \right]$$

- Coefficients are part of the tech files (Parasitic Tables)
- Layer thickness t is normally multiplied into the coefficient
- Foundry tables give constant coefficient referenced to $s = 1$

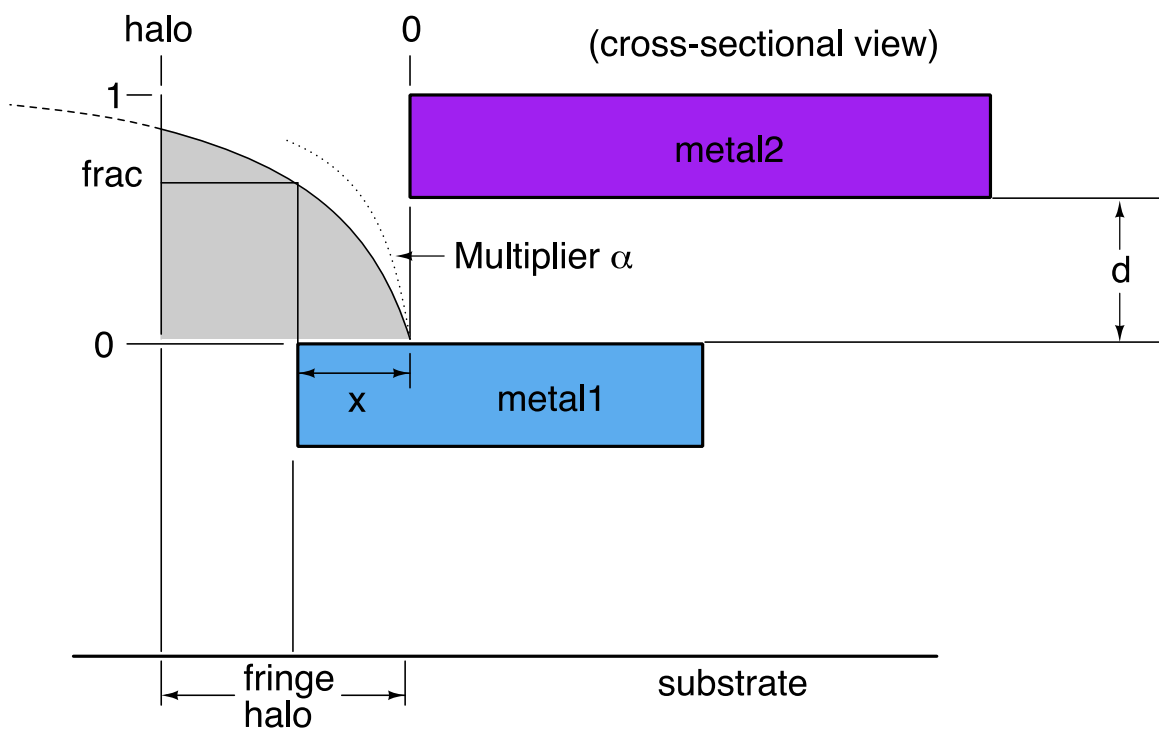
4.2.1.3 Overlap Capacitance



4.2.1.4 Fringe Capacitance

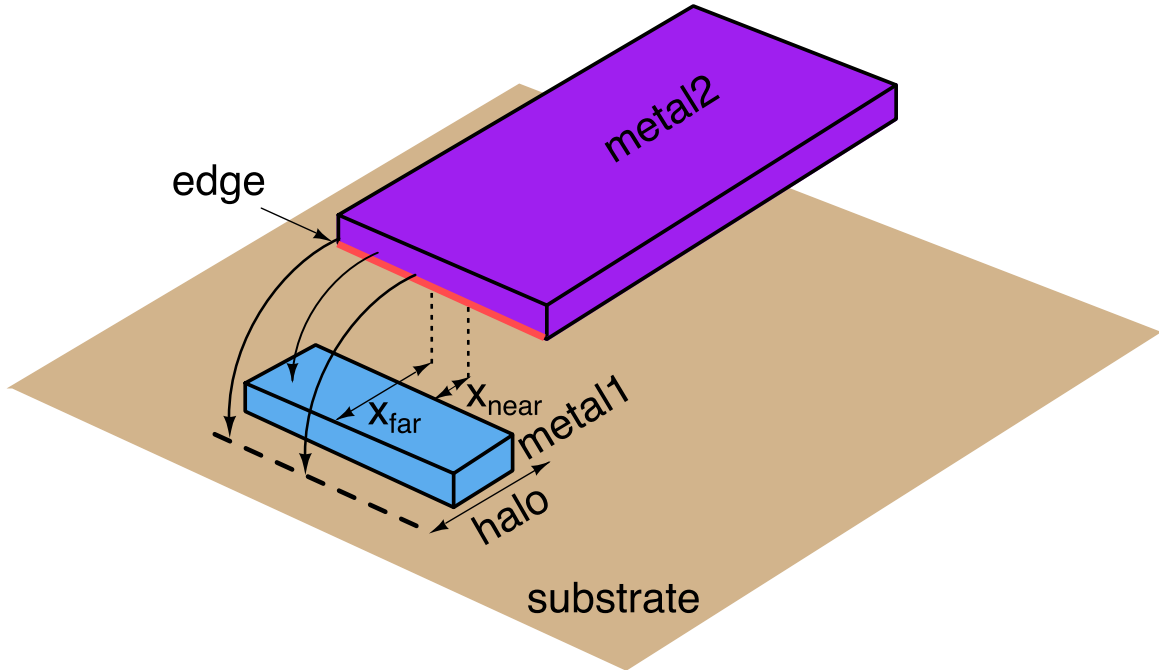


- Causing sidewall (its bottom edge depicted red)
- Assume: Field is bounded by fringe halo (e.g. $8\ \mu m$ away from edge)
- Fractions of fringe goes to metal1 or substrate



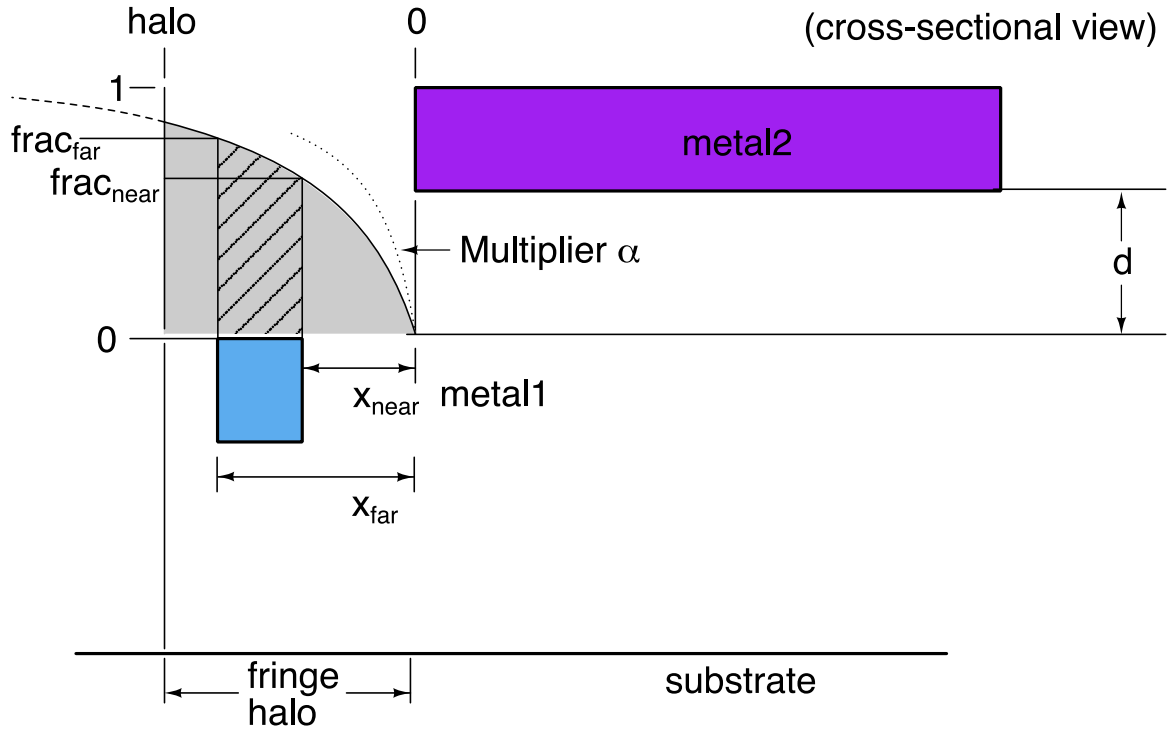
- Multiplier α (comes from tech files: overlap table)

- determines how quickly fringe capacitance drops with increasing distance
- α is related to distance d between layers
- α is proportional to $C_{overlap_{metal1 \leftrightarrow metal2}} = \frac{\epsilon_{si} * K}{d} * \text{area}$
(for fixed value of area $1 \mu m^2$)
- Fringe Fractions:
 - $\text{frac}_{metal1} = \frac{2}{\pi} * \text{atan}(\alpha_{metal2 \rightarrow metal1} * x)$
 - $\text{frac}_{sub} = \frac{2}{\pi} * \text{atan}(\alpha_{metal2 \rightarrow sub} * (\text{halo} - x))$
 - $\frac{2}{\pi}$ is multiplied because of scaling to interval $[0.0, 1.0]$, as $\text{atan}(\infty) = \frac{\pi}{2}$
- Overlap capacitance:
 - $C_{overlap} = \frac{\epsilon_{si} * K}{d} * \text{area}$ (with area = $1 \mu m^2$)
- Coupling capacitance $metal1 \leftrightarrow metal2$:
 - $\alpha_{metal1 \leftrightarrow metal2} = \alpha_{scalefac} * C_{overlap_{metal1 \leftrightarrow metal2}}$
 - $\text{frac}_{metal1} = \frac{2}{\pi} * \text{atan}(\alpha_{metal1 \leftrightarrow metal2} * x)$
 - effective length = edge length * frac_{metal1}
 - $C_{fringe_{metal2 \rightarrow metal1}} = \text{effective length} * C_{sideoverlap_{metal2 \rightarrow metal1}}$
- Coupling capacitance $metal1 \leftrightarrow sub$:
 - $\alpha_{metal1 \leftrightarrow sub} = \alpha_{scalefac} * C_{overlap_{metal1 \leftrightarrow sub}}$
 - $\text{frac}_{sub} = \frac{2}{\pi} * \text{atan}(\alpha_{metal1 \leftrightarrow sub} * (\text{halo} - x))$
 - effective length = edge length * frac_{sub}
 - $C_{fringe_{metal2 \rightarrow sub}} = \text{effective length} * C_{sideoverlap_{metal2 \rightarrow sub}}$



- Partial side overlap

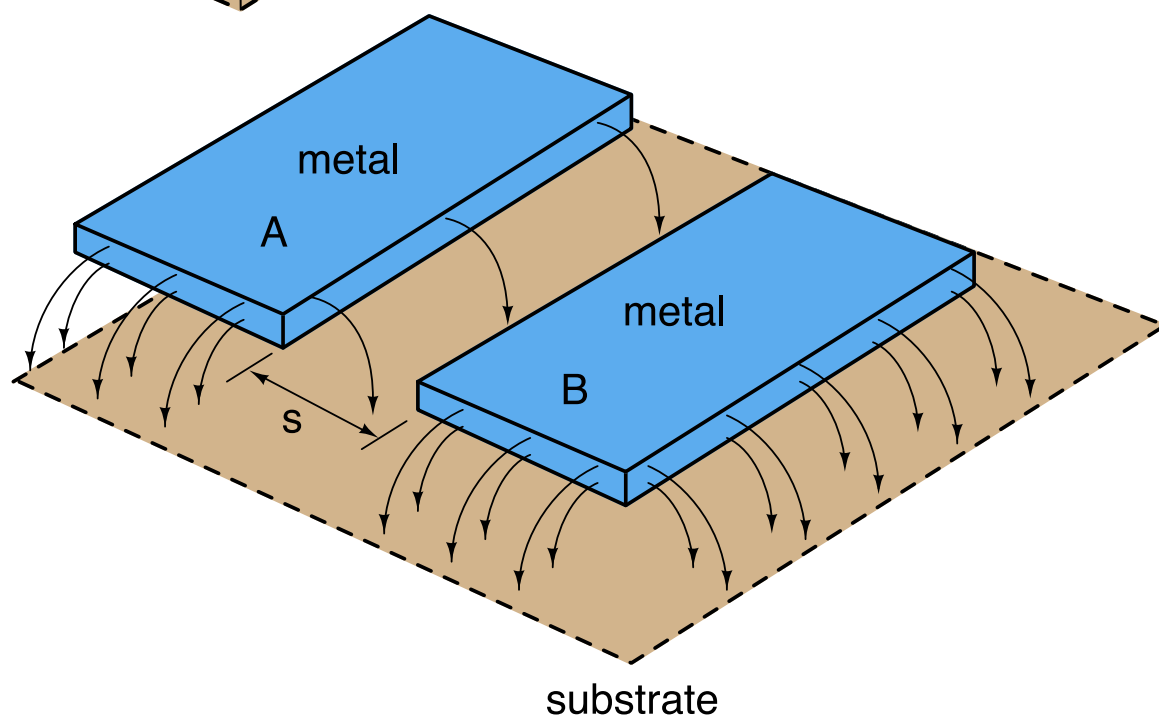
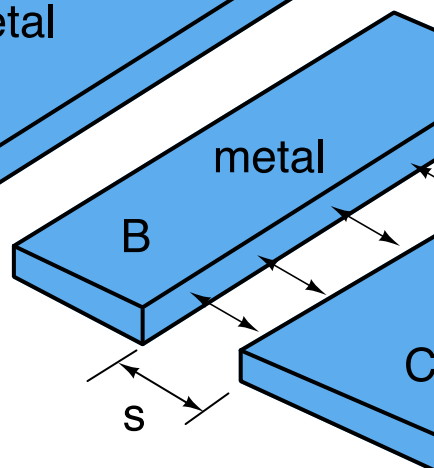
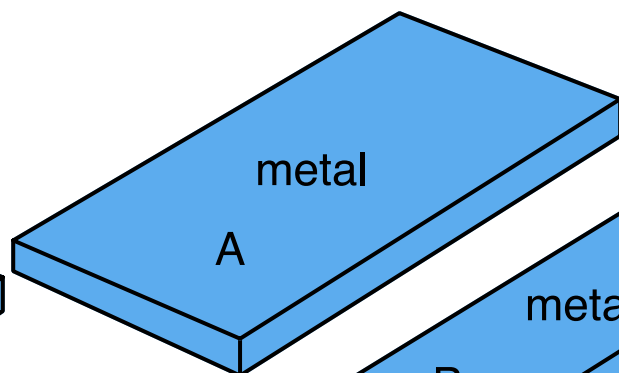
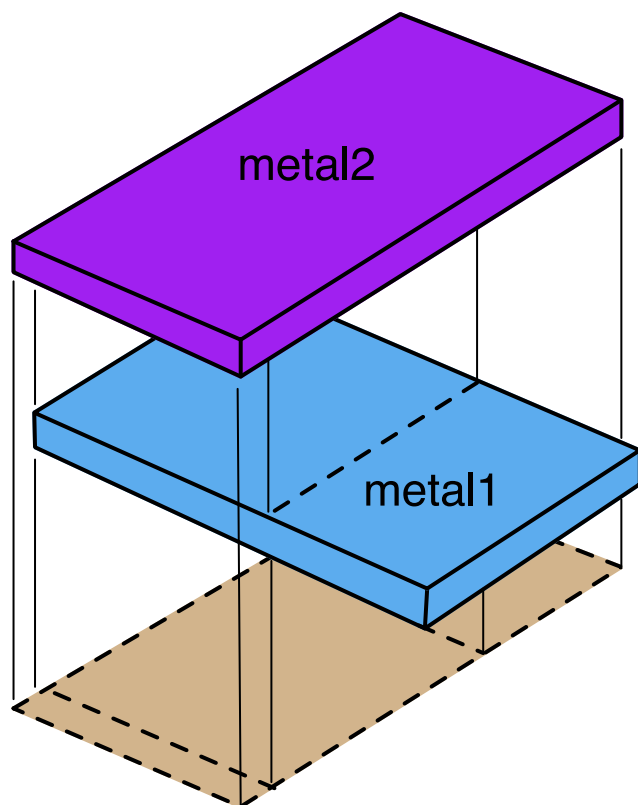
- In case there is only a partial side overlap, the non-existing near fraction is subtracted from the far fraction
- *metal1* wire is offset, starts at x_{near}
- *metal1* ends at x_{far}

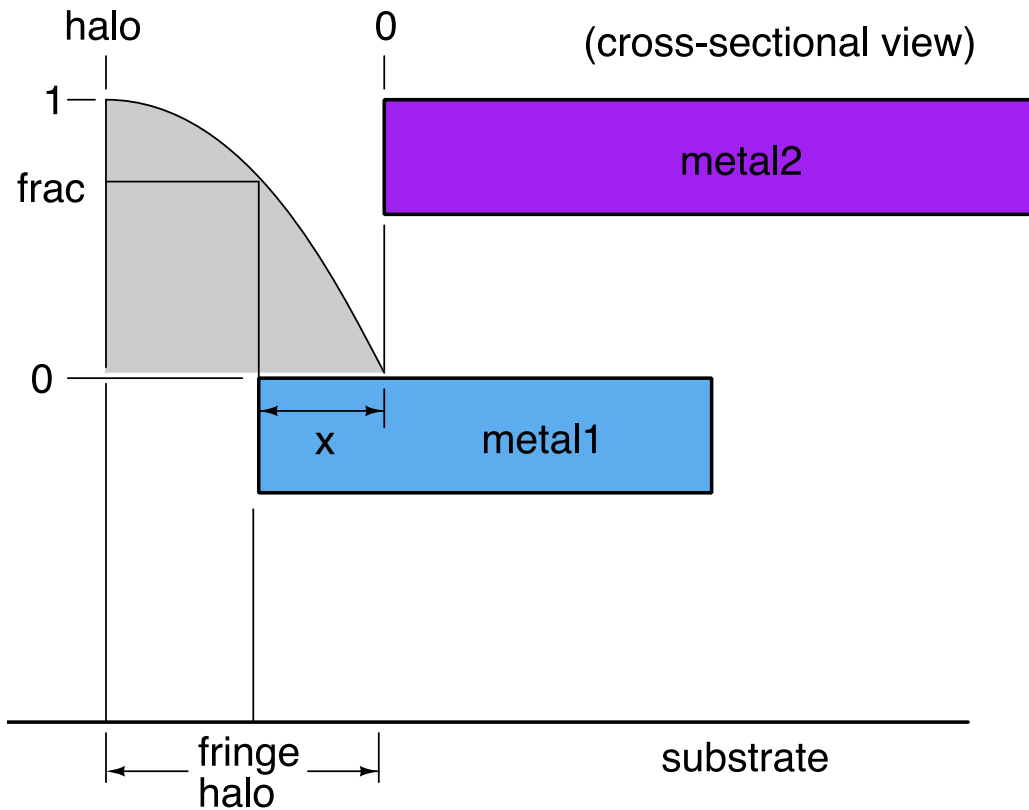


- $\text{frac}_{near} = \frac{2}{\pi} * \text{atan}(\alpha * x_{near})$
- $\text{frac}_{far} = \frac{2}{\pi} * \text{atan}(\alpha * x_{far})$
- $\text{frac} = \text{frac}_{far} - \text{frac}_{near}$

4.2.1.5 Shielding Effects

Type	Shielding To Substrate	Between layers	On same layer
Overlap shielding			
Sidewall shielding			
Lateral fringe shielding			
Vertical fringe shielding			





4.3 2.5D Engine

Field solvers are precise, yet slow, they are useful to obtain a golden reference. For most use cases, a faster engine is desirable. The KPEX 2.5D Engine intends to implement the concepts and formulas of MAGIC (see Section 4.2.1) in a way that is best suited to the KLayout API.

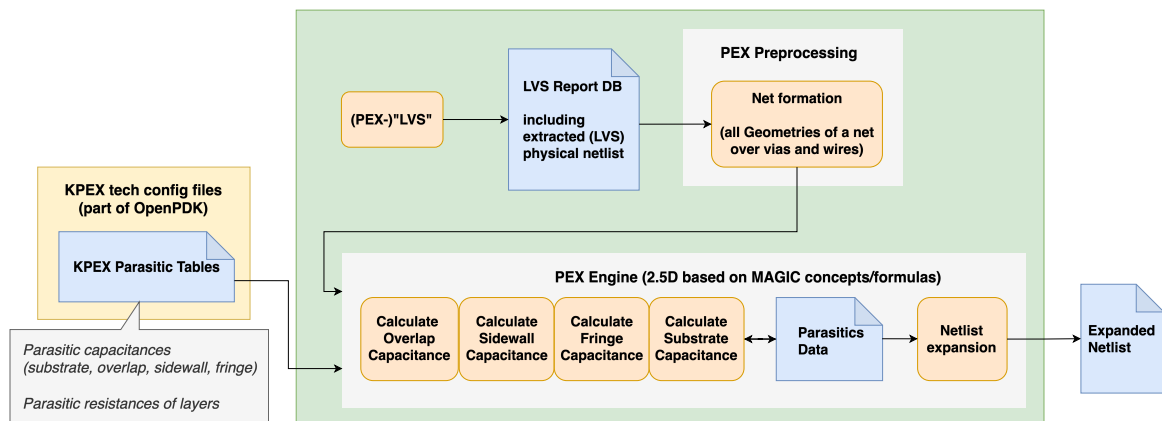


Figure 6: KPEX 2.5D Engine