Palace FEM workflow for IHP gds2palace, October 2025

Volker Mühlhaus, volker@muehlhaus.com

Document version: 2025-10-23

Contents

About this workflow

Palace, for **PA**rallel **LA**rge-scale **C**omputational **E**lectromagnetics, is an open-source, parallel finite element code for full-wave 3D electromagnetic simulations. It can be scaled from single computer to large high performance simulation clusters and cloud-based computing.

https://awslabs.github.io/palace/stable/

https://aws.amazon.com/de/blogs/quantum-computing/aws-releases-open-source-software-palace-for-cloud-based-electromagnetics-simulations-of-quantum-computing-hardware/

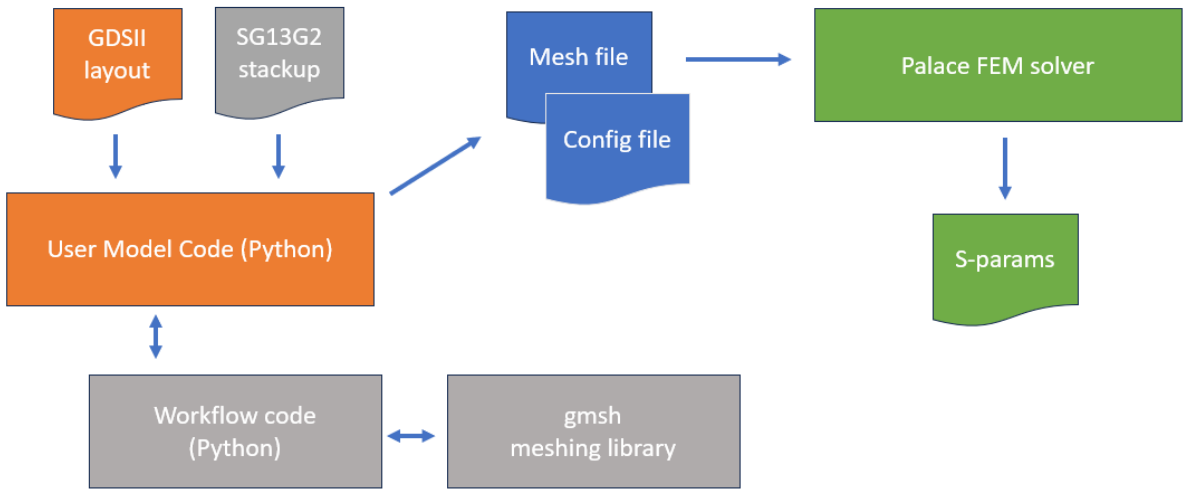
The gds2palace workflow enables RFIC FEM simulation using Palace from GDSII layout files.

Workflow

In this document, the status of the IHP workflow developed for AWS Palace FEM solver is documented. The picture below gives an overview of the workflow.

Two files must be provided by the user: the **layout in GDSII format** and a **simulation model script** in Python. This simulation script references the technology stackup file (XML format) and calls functions from the workflow code to create two output files for Palace: the **simulation mesh file** which is built from geometries and stackup, and the **simulation control file** in *.json format which defines material settings and simulation settings.

Having these two files, the user can now run the FEM simulation in Palace. The simulation model script can start an external command to start simulation, or the user can run Palace on a platform of his choice. This way, the files for Palace can be created on any desktop computer, and the Palace simulation can be done on the same computer or on a very different system. This gives many options to scale Palace simulation power as needed, using the exact same Palace input data from our workflow.



Required software and Python modules

This workflow creates AWS Palace model files (mesh file + config file).

The gds2gmsh workflow described here requires these Python modules:

- gdspy (version 1.6.13 or later recommended)
- gmsh
- scikit-rf

To use the same file locations as described in this manual, you can create a Python venv named “palace” in your home directory:

```
cd ~
python3 -m venv ~/venv/palace
source ~/venv/palace/bin/activate
pip install gdspy
pip install gmsh
pip install scikit-rf
```

This virtual environment can now be used to run gds2palace and the scripts that postprocess the Palace simulation results. To activate this Python environment on the command line, you can run

```
source ~/venv/palace/bin/activate
```

or you can run a Python file using the python executable from that directory.

Installing AWS Palace

It is assumed that you have already installed AWS Palace to run the simulation from the model files. Installing Palace is described here: https://awslabs.github.io/palace/stable/install/

For development of this workflow, Palace was installed using the Singularity/Apptainer installation method. This was rather simple and straightforward, even with no knowledge about container usage. The resulting apptainer file palace.sif can be integrated very easily in a Linux system like the Ubuntu 24.04 system used here, and can then be moved to other Linux machines using simple copy of the container file. Highly recommended! Some more description on this configuration can be found in this chapter: Running the model code to create Palace input files.

But of course, you can also use one of the other installation methods described on the AWS Palace web site. The gds2palace workflow does not change, it only creates the input files for Palace and does not care how you installed Palace, or on what platform you run the actual Palace simulation from these model files.

Quick tour

This chapter will give a very brief overview of the workflow usage, without going into the details.

To get started, we need a directory that contains:

- GDSII layout file to be simulated
- XML stackup file
- Workflow modules directory gds2gsmh

In addition, we need the simulation model code in Python, which brings the workflow to life. Below are the most important code sections that need to be configured by the user:

Simulation model: Input files

In this section, the user specified the GDSII data source and the stackup file.

```
# ==================== input files and path settings ==================

gds_filename = "line_simple_viaport.gds"    # geometries
XML_filename = "SG13G2_nosub.xml"           # stackup

# preprocess GDSII for safe handling of cutouts/holes?
preprocess_gds = False
```

Simulation model: Simulation control

In this section, the user needs to specify frequency range and mesh settings. There are additional optional settings that can be applied, as discussed later in this document.

```
settings['unit']   = 1e-6  # geometry is in microns
settings['margin'] = 50    # distance in microns from GDSII geometry boundary to simulation boundary

settings['fstart']  = 0e9
settings['fstop']   = 100e9
settings['fstep']   = 2.5e9

settings['refined_cellsize'] = 2  # mesh cell size in conductor region
settings['cells_per_wavelength'] = 10   # how many mesh cells per wavelength, must be 10 or more

settings['meshsize_max'] = 70  # microns, override cells_per_wavelength
settings['adaptive_mesh_iterations'] = 0
settings['z_thickness_factor'] = 0.33
```

Users who are familiar with the IHP openEMS workflow will notice many similarities, although some implementation details are different.

One important difference is that FEM simulates one frequency after another, so frequency settings have an effect on total simulation time. However, Palace is configured to use an adaptive frequency sweep, which will EM simulate only the minimum required number of frequencies across the band, and then create all other frequency points using interpolation. This adaptive sweep is enabled by default in our workflow.

Another important difference is the use of “refined_cellsize” to create the mesh: the gmsh meshing engine will always create a mesh based on all geometry detail, no matter how small it is, and then apply an additional mesh refinement based on “refined_cellsize” along the metal edges.

Simulation model: Ports

The third important user input are port definitions.

Similar to the IHP openEMS workflow, **ports are created based on polygons from the GDSII file**, located on special layers that are not part of the IHP layer table. Each port needs to use a different source layer in the GDSII file.

The port mapping defines the port number and port impedance, and then maps the port geometry from the special GDSII input layer (usually 201 and above) to actual IHP technology layers. For the vertical ports shown here, we have from_layername and to_layername. For in-plane ports, we would have target_layername instead. Finally, the port direction is required to specify vertical ports or in-plane ports and their direction/polarity.

```
simulation_ports = simulation_setup.all_simulation_ports()
# instead of in-plane port specified with target_layername, we here use via port specified with from_layername and to_layernam
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1, voltage=1, port_Z0=50,
                                                           source_layernum=201,
                                                           from_layername='Metal1', to_layername='TopMetal2', direction='z'))
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=2, voltage=1, port_Z0=50,
                                                           source_layernum=202,
                                                           from_layername='Metal1', to_layername='TopMetal2', direction='z'))
```

For this Palace workflow, the voltage parameter is not supported yet by the Palace solver. Palace only supports opposite polarity by reversing the port direction. **But here in our workflow, we use this voltage parameter to specify if a port is active: ports with voltage=0 will not be excited.**

It must be noted that Palace behaves differently from other FEM solvers: to get the full S-matrix, we need to run all port excitations, one after another. This only happens if all port voltages are not zero. Only then, the full S-parameter output file can be created.

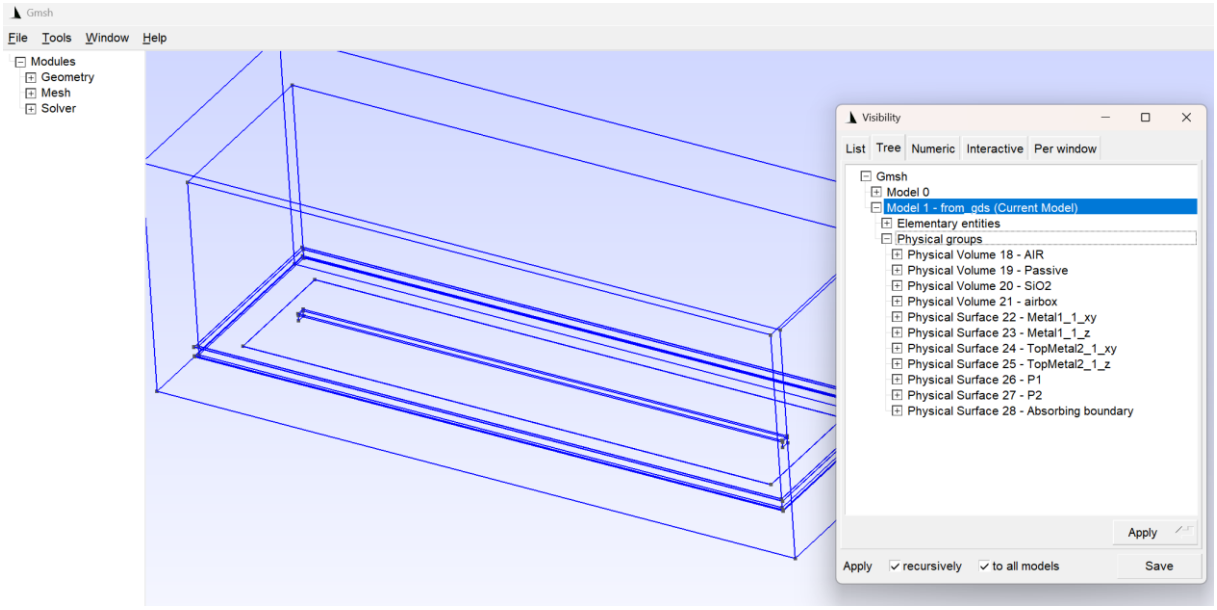
ATTENTION: If any port excitation is zero, that “zero voltage” port will not be excited and that row in the S-parameter file will be padded with zeros. For example, if we simulate an 8-port circuit and only excite port 1, the S-parameter output will have valid results for S11, S21, … S81 but all other values will be zero. This can be useful to quickly simulate one specific path in the model.
To get the full S-parameter file, all ports must be defined with non-zero voltage, so that all port excitations are simulated, one after another.

PORT SHAPE IN GDSII: The workflow creates lumped ports in Palace, which need to be 2D sheets. This means that for vertical ports (via ports), you should draw a zero-width box in GDSII, resulting in a vertical 2D sheet with no width. If you define a vertical port from a box with finite xy area, the Palace port will be created as a vertical 2D sheet along the center line of that 2D box.

Composite ports (multiple EM ports grouped into one port in the final output file) are not yet supported by this workflow. All ports with non-zero voltage are simulated one after another, resulting in n-port S-parameters.

Running the model code to create Palace input files

The simulation model file (Python code) can be run on the command line. After reading and processing the input files, a 3D viewer comes up and shows the resulting 3D model. This viewer is the graphical interface of the gmsh meshing library, and provides many options for inspection of the model. At this point, the model is not meshed yet, so that we can see the raw geometries.



To see the structure of the 3D model, you can go to Tools > Visibility.

In the screenshot, you can see that dielectric boxes (SiO2, Passive, AIR etc) have been added around the GDSII polygons, with an offset value in xy direction from the “margins” parameter in the model file. Around that, we have another layer of air on all six sides, using the same margins value.

Metals have been created as surfaces. All polygons on each layer are merged, if possible, and then each of the resulting polygons is created as a separate surface. To be more precise, we have two surfaces for each polygon: the horizontal (xy) surfaces and the vertical (z) surfaces are assigned to different groups, for reasons explained later in this document.

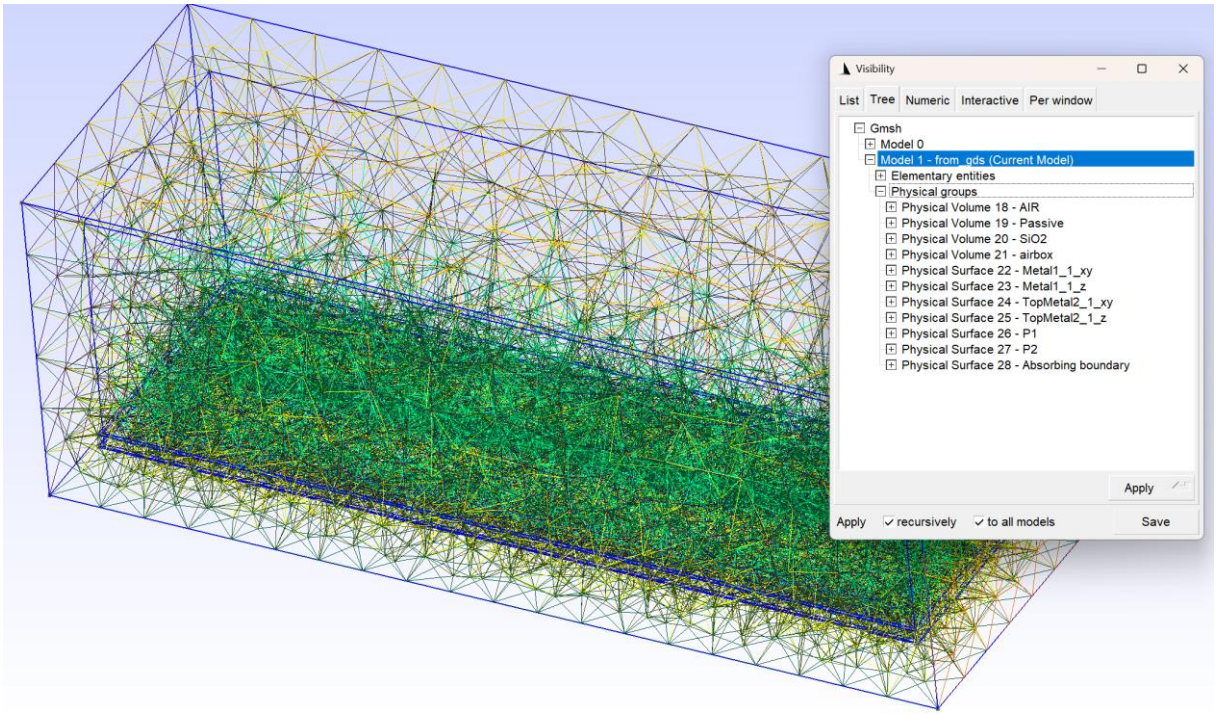
You can also see two ports P1 and P2 created as surfaces.

The outer simulation boundary is this example is a surface “Absorbing boundary” that is defined as absorbing boundary in the Palace config file.

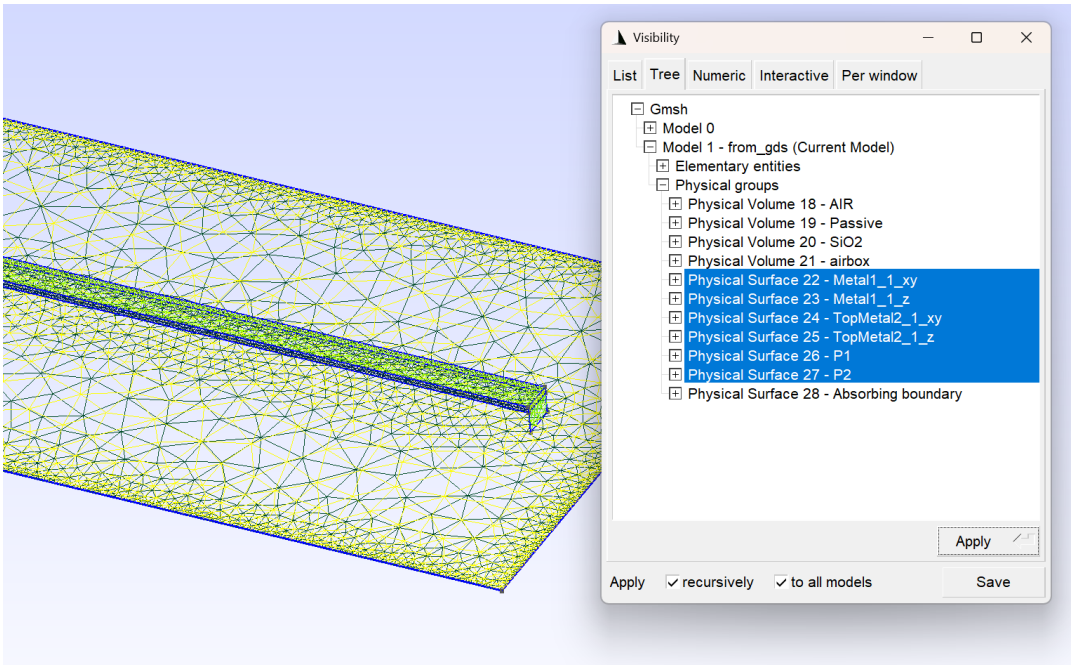
When we are done with inspecting the model (which is optional, no user action is required!), the gmsh viewer window can be closed to proceed with meshing.

After closing the gmsh geometry preview, the simulation model script will mesh these geometries, which can take a while and will show lots of status information on the command line.

When meshing is completed, the gmsh 3D viewer will be displayed again, showing the overall mesh.

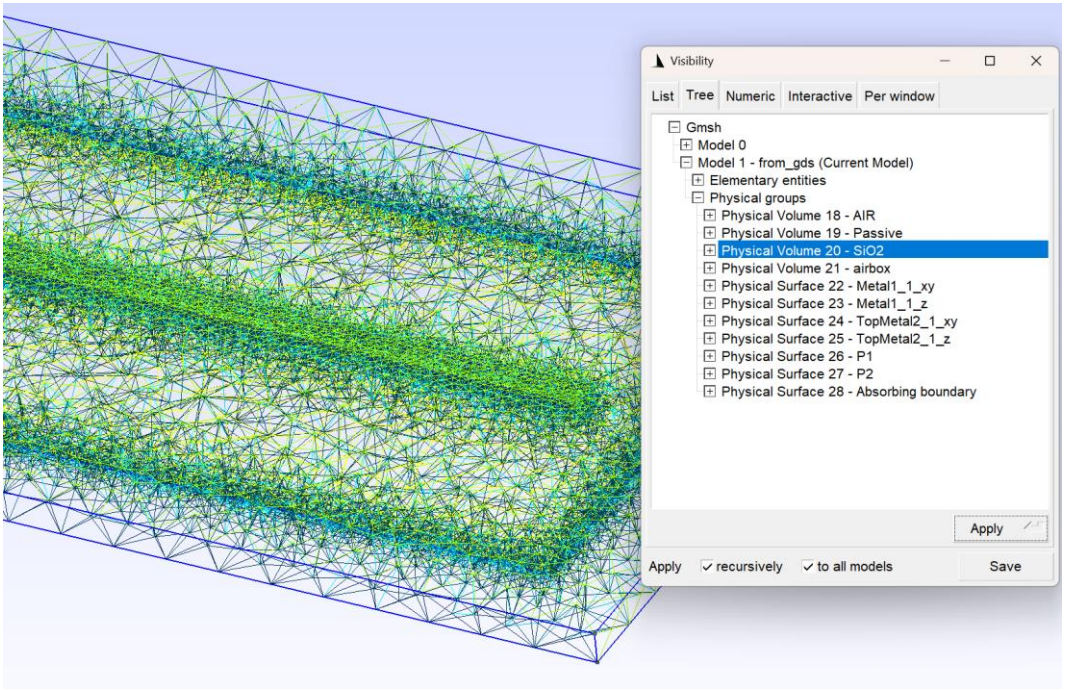


This is too complex to see anything, but we can now go to Tools > Visibility and select one or more groups to be displayed:



This shows the mesh at the conductor surfaces of Metal1, TopMetal1 and the vertical ports.

We can also display the meshed oxide, which has a lot of detail because the metal layers reside inside this volume. The metals have been “cut out” from the dielectric layer, and we see the mesh refinement around the conductor edges.



When you are done with inspection of the model (which is optional, no user action is required!), the gmsh viewer window can be closed to finish model generation.

The workflow code has now created the required file to start Palace: the simulation control file config.json and the mesh file.

config.json

Datei Bearbeiten Ansicht

```
{
    "Problem": {
        "Type": "Driven",
        "Verbose": 3,
        "Output": "output/palace_line_viaport"
    },
    "Model": {
        "Mesh": "palace_line_viaport.msh",
        "L0": 1e-06,
        "Refinement": {}
    },
    "Solver": {
        "Driven": {
            "MinFreq": 0.1,
            "MaxFreq": 100.0,
            "FreqStep": 2.5,
            "SaveStep": 0,
            "AdaptiveTol": 0.01
        },
        "Linear": {
            "Type": "Default",
            "KSPType": "GMRES",
            "Tol": 1e-06,
            "MaxIts": 400
```

Running Palace FEM simulation from our input files

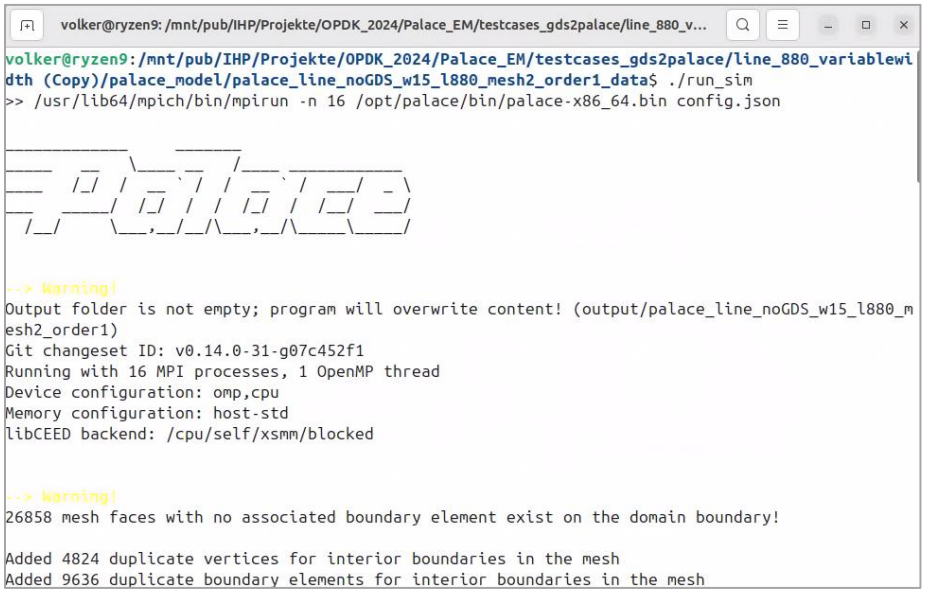
To simplify running the solver, in addition to mesh file and config file, a script file was also created named “run_sim”.

This starts a script “run_palace” where you can define in detail how to run Palace. It is recommended that you place this run_palace script in your PATH, configured for the actual machine where you want to run Palace.

For workflow development and test, Palace was installed into an apptainer container, as documented in the Palace documentation in chapter “**Build using Singularity/Apptainer**”.

The script “run_palace” was then configured to start Palace inside the palace_014.sif container, which is located in the user’s home directory, and passes one command line argument (the config.json file). The additional parameter *-np 16* tells palace to run using 16 threads. This will partition the simulation domain into 8 pieces, each running in a separate process, and the combine results into one output directory.

Running Palace is not very spectacular, the simulation progress is shown in the terminal window.



When “run_palace” is finished, Palace output files are created in the “output” directory below the simulation model directory. S-parameters are in *.csv file format, which we need to convert now.

To do that conversion, the run_sim script where we started simulation with “run_palace” executes another command “combine_snp”.

combine_snp is a script that searches for Palace s-parameter files (port-S.csv) and converts them to Touchstone file format. If the model was simulated for selected port excitations only, missing rows in the S-parameter file will be padded with zeros.

It should be noted that Palace frequency domain simulation cannot go down to 0 Hz, and the workflow script will replace any 0 Hz start frequency by a low value like 1 GHz.

If simulation results include such low frequency data, combine_snp will create an additional S-Parameter file with suffix “_dc”, with a DC value extrapolated from the EM simulated data.[1] Always check that DC extrapolated dataset carefully before use!

This ends our quick tour of using the gds2palace workflow. There are many additional settings available, which will be described in detail in the next chapters.

[1] combine_snp is a Python script created for this workflow. DC extrapolation in this script is powered by Python library scikit-rf.

Simulation model file in detail

Input files

Typical simulation models require two input files: GDSII geometries and XML stackup.

```
# ==================== input files and path settings ======================

gds_filename = "BM_Ardavan_Rahimian_with_ports.gds"   # geometries
XML_filename = "SG13G2_nosub.xml"          # stackup
```

Two settings are available to control processing of the GDSII geometries:

```
# preprocess GDSII for safe handling of cutouts/holes?
preprocess_gds = True

# merge via polygons with distance less than .. microns, set to 0 to disable
merge_polygon_size = 0
```

pre_process_gds must be enabled if the layout includes any cutouts (holes) or other self-intersecting polygon boundaries. This will split such polygons into smaller entities, which can be processed properly. Without that preprocessing, mesh generation of layout with holes would fail with error messages like “Exception: Curve loop is not closed”

merge_polygon_size applies to layers which are declared as *Type=”via”* in the XML stackup file. When enabled with a non-zero value, the code will merge polygons on these layers with the given distance. This is how it works: in the GDSII reader, polygons on this layer will be oversized by half the given value, then all overlapping polygons on that layer will be merged, then undersized by half the given value.
For via arrays that are oriented along the xy-axis, this will give the resulting bounding box if the maximum via spacing is no larger than the given value.

Settings

Settings for meshing and simulation control are implemented as a Python dictionary, which is passed to the simulation_setup.create_palace function.

Some settings are required, other are optional with a meaningful default value.

These settings are always required:

settings['unit']	Unit of values in mesh, typically 1E-6
settings['margin']	Oversize of dielectrics from bounding box of drawing in xy plane
settings['fstart']	Start frequency in Hz
settings['fstop']	Stop frequency in Hz
settings['fstep']	Frequency step in Hz for output, not all of them need to be EM simulated due to adaptive frequency sweep
settings['refined_cellsize']	Target mesh size at polygon edges

There are additional **optional** settings to specify fixed discrete frequencies, which you can use in addition to fstart/fstop and fstep, or instead of fstart/fstop and fstep:

settings['fpoint']	Discrete frequency/frequencies, values enclosed in [ ] Example: settings["fpoint"] = [10e9, 15e9]
settings['fdump']	Same as fpoint, but Palace is configured to write a field dump for Paraview at this frequency/these frequencies. Example: settings["fdump"] = [10e9]

These other settings are optional:

	Meaning	**Default value**
settings['cells_per_wavelength']	Calculated at highest frequency, value must be 10 or more	10
settings['meshsize_max']	Maximum mesh size limit, in addition to cells/λ	70
settings['air_around']	Other spacing of air around dielectrics	same as margin
settings['substrate_refinement']	Extra mesh refinement into substrate	False
settings['adaptive_sweep']	Enable adaptive frequency sweep	True
settings['adaptive_mesh_iterations']	Iterations for adaptive mesh refinement	0
settings['save_adaptive_mesh']	Save mesh file from adaptive iteration for possible re-use	False
settings['save_gmsh_unrolled']	Also save gmsh geometry file without meshing, for later inspection	False
settings['z_thickness_factor']	Factor for metal thickness value on conductor side walls[2]	0.33
settings['boundary']	List with 6 values for boundary at xmin,xmax,ymin,ymax,zmin,zmax Values can be PML, PEC or PMC	['PML','PML','PML', 'PML','PML','PML']
settings['order']	Order of basis function for FEM solver	2
settings['no_gui']	Run script without showing gmsh user interface, useful for automated processing	False

[2] See chapter on metal loss at low frequency, where skin depth is larger than metal thickness

Port configuration

Similar to the IHP openEMS workflow, **ports are created based on polygons from the GDSII file**, located on special layers that are not part of the IHP layer table. Each port needs to use a different source layer in the GDSII file.

The port mapping defines the port number and port impedance, and then maps the port geometry from the special GDSII input layer (usually 201 and above) to actual IHP technology layers. For the vertical ports shown here, we have from_layername and to_layername. For in-plane ports, we would have target_layername instead. Finally, the port direction is required to specify vertical ports or in-plane ports and their direction/polarity.

```
simulation_ports = simulation_setup.all_simulation_ports()
# instead of in-plane port specified with target_layername, we here use via port specified with from_layername and to_layernam
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1, voltage=1, port_Z0=50,
                                                           source_layernum=201,
                                                           from_layername='Metal1', to_layername='TopMetal2', direction='z'))
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=2, voltage=1, port_Z0=50,
                                                           source_layernum=202,
                                                           from_layername='Metal1', to_layername='TopMetal2', direction='z'))
```

For this Palace workflow, the voltage parameter is not supported yet by the Palace solver. Palace only supports opposite polarity by reversing the port direction. **But here in our workflow, we use this voltage parameter to specify if a port is active: ports with voltage=0 will not be excited.**

It must be noted that Palace behaves differently from other FEM solvers: to get the full S-matrix, we need to run all port excitations, one after another. This only happens if all port voltages are not zero. Only then, the full S-parameter output file can be created.

ATTENTION: If any port excitation is zero, that “zero voltage” port will not be excited and that row in the S-parameter file will be padded with zeros. For example, if we simulate an 8-port circuit and only excite port 1, the S-parameter output will have valid results for S11, S21, … S81 but all other values will be zero. This can be useful to quickly simulate one specific path in the model.
To get the full S-parameter file, all ports must be defined with non-zero voltage, so that all port excitations are simulated, one after another.

PORT SHAPE IN GDSII: The workflow creates lumped ports in Palace, which need to be 2D sheets. This means that for vertical ports (via ports), you should draw a zero-width box in GDSII, resulting in a vertical 2D sheet with no width. If you define a vertical port from a box with finite xy area, the Palace port will be created as a vertical 2D sheet along the center line of that 2D box.

Composite ports (multiple EM ports grouped into one port in the final output file) are not yet supported by this workflow. All ports with non-zero voltage are simulated one after another, resulting in n-port S-parameters.

Port parasitics in results, lumped port vs. wave port: This workflow creates lumped ports, which introduce some physical length into the simulation model, leading to extra path length (→ inductance) in results. Other 3D volume meshing EM tools behave the same for lumped ports. If port size is not small compared to the device under test, one possible solution is to estimate these parasitics and remove them in postprocessing (or result use) by cascading negative inductance or negative length at each port.

Filenames and flow control

If you want to have additional control over the names of files and directories that are created by the workflow, have a look at this section at the beginning of the simulation model file:

```
# get path for this simulation file
script_path = utilities.get_script_path(__file__)

# use script filename as model basename
model_basename = utilities.get_basename(__file__)

# set and create directory for simulation output
sim_path = utilities.create_sim_path (script_path,model_basename)
print('Simulation data directory: ', sim_path)

# change path to models script path
modelDir = os.path.dirname(os.path.abspath(__file__))
os.chdir(modelDir)
```

For example, if you want the customize the model name, to reflect settings used for meshing, you can define these values as variables and append that to the model_basename variable.

```
mesh  = 5  # define here, so that we can add this information to output filenames
order = 3  # define here, so that we can add this information to output filenames

XML_filename = "SG13G2_nosub.xml"          # stackup

# preprocess GDSII for safe handling of cutouts/holes?
preprocess_gds = False

# merge via polygons with distance less than .. microns, set to 0 to disable via merging.
merge_polygon_size = 0

# get path for this simulation file
script_path = utilities.get_script_path(__file__)

# use script filename as model basename, with parameters for suffix
model_basename = utilities.get_basename(__file__) + '_mesh' + str(mesh) + '_order' + str(order)

# set and create directory for simulation output
sim_path = utilities.create_sim_path (script_path,model_basename)
print('Simulation data directory: ', sim_path)

# change path to models script path
modelDir = os.path.dirname(os.path.abspath(__file__))
os.chdir(modelDir)

# ======================= simulation settings ==============================

settings = {}

settings['unit']   = 1e-6  # geometry is in microns
settings['margin'] = 50    # distance in microns from GDSII geometry boundary to simulation boundary

settings['fstart']  = 0e9
settings['fstop']   = 100e9
settings['fstep']   = 2.5e9

settings['refined_cellsize'] = mesh  # mesh cell size in conductor region
settings['cells_per_wavelength'] = 10   # how many mesh cells per wavelength, must be 10 or more

settings['meshsize_max'] = 70  # microns, override cells_per_wavelength
settings['adaptive_mesh_iterations'] = 0
settings['z_thickness_factor'] = 0.33
settings['order'] = order
```

If you want the script to create the Palace files without showing the mesh in gmsh 3D viewer, this is also possible.

You can use the optional *settings['no_gui']* and either set this to *False* (always run without graphical user interface) or you can check for an optional command line parameter passed to the simulation model script.

```
settings['nogui'] = ('nogui' in sys.argv)  # check if nogui specified on command line
```

In this case, running the model as

```
python mymodel.py
```

would run with 3D mesh viewer, or running the model as

```
python mymodel.py nogui
```

would run without the 3D viewer. Maybe this gives some idea what customization is possible.

Examples

This chapter gives examples on configuration settings, especially mesh control, and their results.

Single microstrip line

The first testcase for meshing is a microstrip line with TopMetal2 over Metal1 ground. Width is 15 µm to achieve 50 Ohm line impedance, length is 880µm.

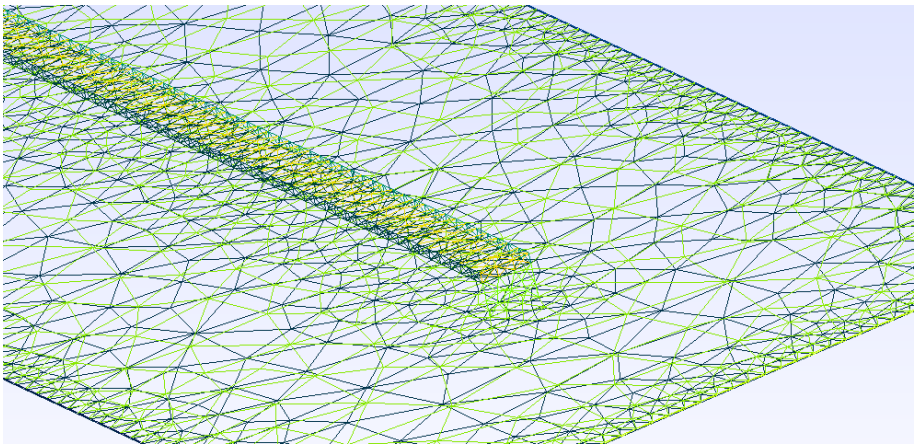
This model is simulated with mesh configurations that vary 2 mesh parameters:

- **'refined_cellsize'** defines the mesh resolution along the conductor edges
- **'order'** defines the order of basis functions, i.e. the degrees of freedom for field variation within one cell

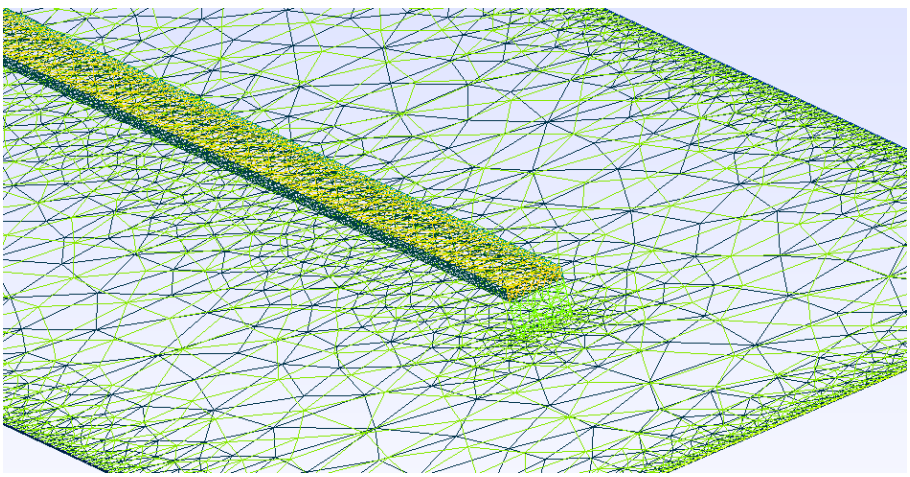
By default, we want to simulate with an order=2, which is a good trade-off between accuracy and speed. However in this simple example, we will check how these two values can be configured for accurate results, and what this does to simulation time.

The basic idea is this: if we have a higher order of basis functions, the fields can be modelled more accurately within each mesh cell, so we need less mesh cells → larger value of refined_cellsize.
If we use first order basis functions, we need higher mesh density to accurately model the field distribution.

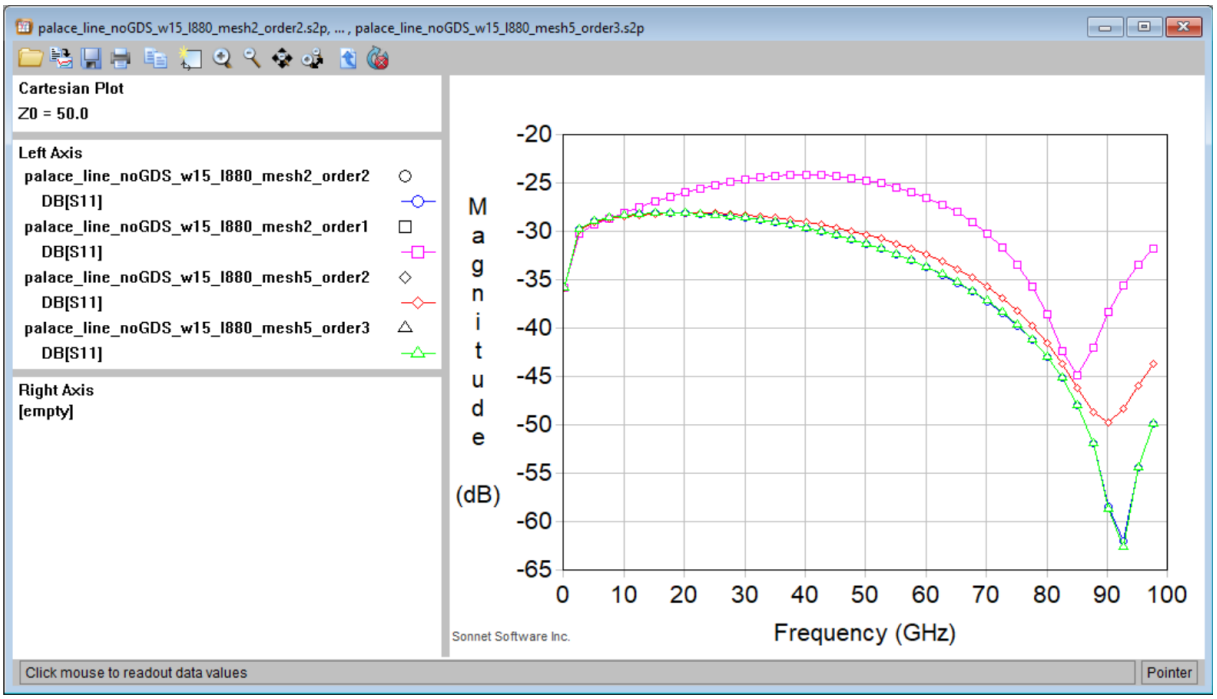
Mesh on conductors at **'refined_cellsize'** = 5:



Mesh on conductors at **'refined_cellsize'** = 2:



These two mesh densities are now simulated with different order of basis function.

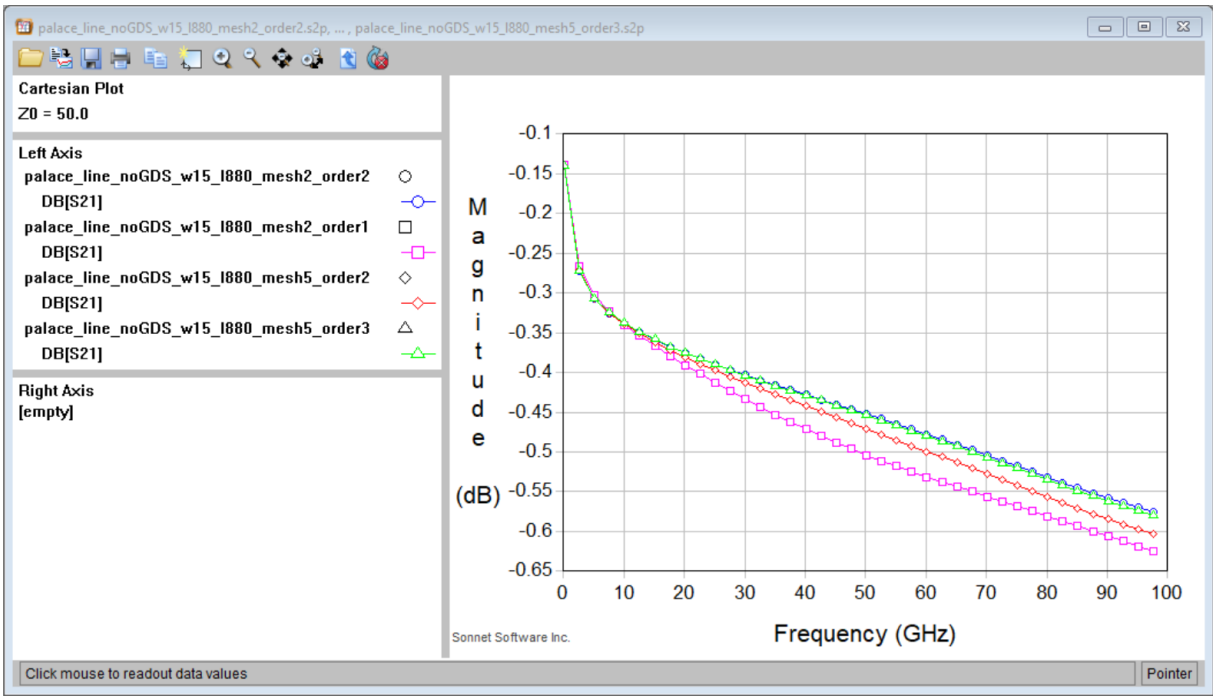


The clear outlier in this return loss plot is the pink curve, simulated with mesh order = 1 at 2µm refined cellsize. This result is different from the other curves obtained with higher mesh order.

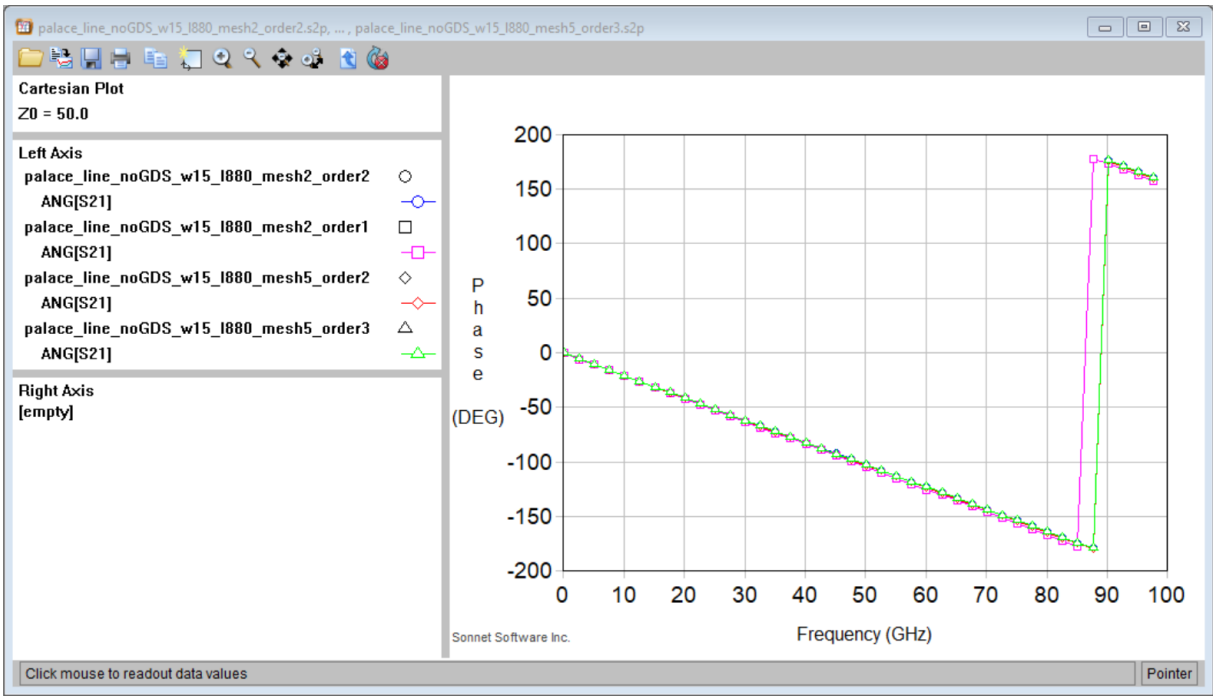
Our default simulation would be the blue curve, obtained with 2µm refined cellsize and order = 2.

Going to a larger value refined_cellsize = 5µm at order = 2 gives slightly different results, shown by the red curve.

If we combine the larger refined_cellsize = 5µm with higher order = 3, to give each cell more degrees of freedom, we get similar results as our baseline (default) simulation: the green and blue curve are visually identical for S11. For S21 transmission, we get a similar result: these two are almost identical.



For phase of S21 we also have similar results: the order = 1 result is a clear outlier, but all other results are very close.



Conclusion: refined_cellsize=2 at order=2 is a good starting point, and for this example we don't need to go any further. Decreasing the mesh resolution to refined_cellsize=5 at order=2 is possible if we can accept some small change in results.

Let's look at simulation times for these 4 different meshing approaches:

	Simulation time total	**Degrees of freedom (from log)**
refined_cellsize=2, order=2	733 s	1640126
refined_cellsize=2, order=1	64 s	321999
refined_cellsize=5, order=2	282 s	699658
refined_cellsize=5, order=3	1967 s	1982661

All simulations were done using adaptive frequency sweep and required 10 discrete frequencies to be EM simulated. Total time in the table also includes interpolation to the specified sweep range.

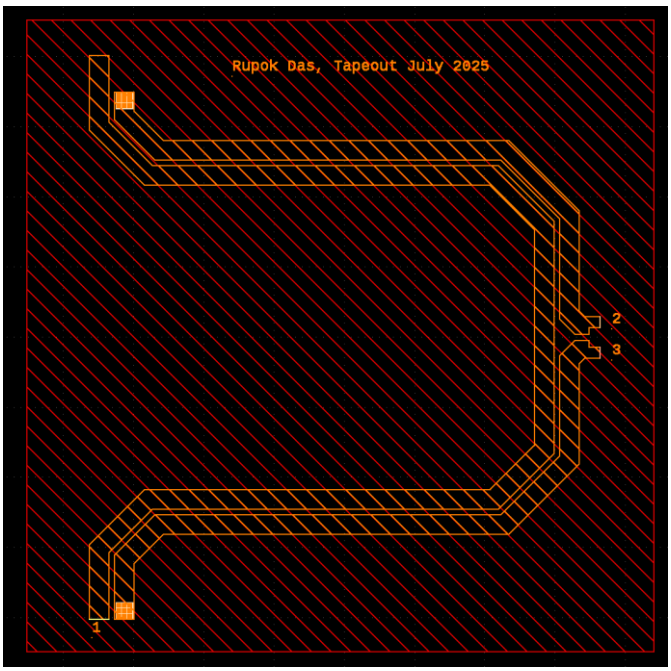
It can be seen that refined_cellsize=2, order=2 is the best choice for accurate data, we get results that are visually identical to refined_cellsize=5, order=3 at less simulation time (733 s total vs. 1967 s total).

In the next chapter, we will go from that baseline setting, and investigate the use of adaptive mesh refinement.

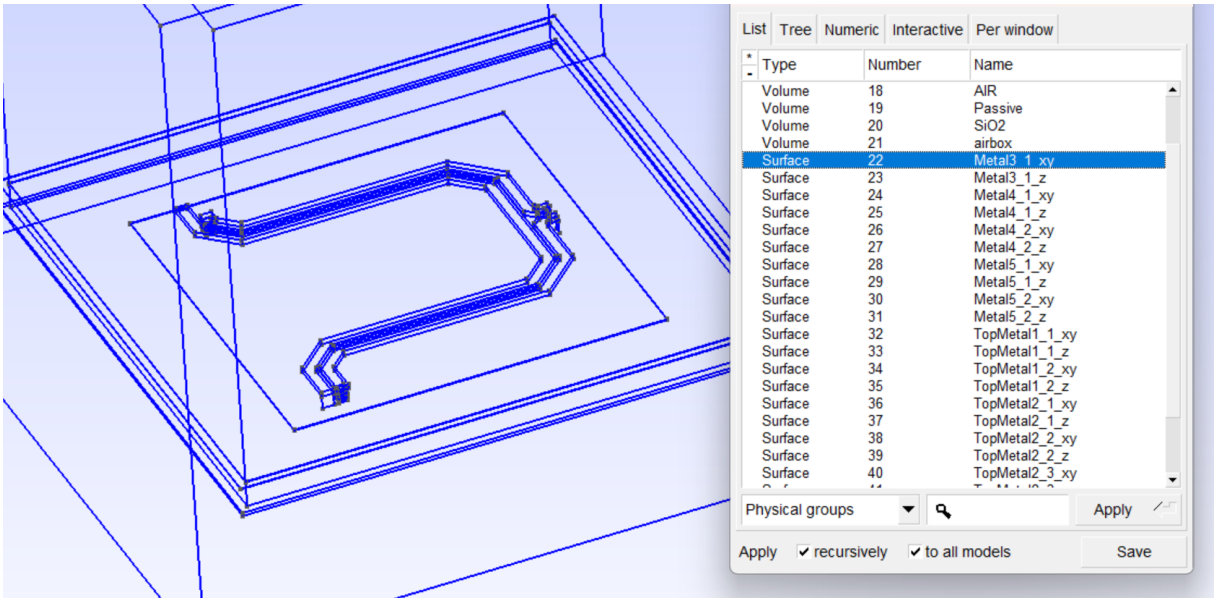
Balun 140-170 GHz

This testcase is a balun from the phase shifter project by Rupok Das, published for the OPDK July 2025 tapeout:

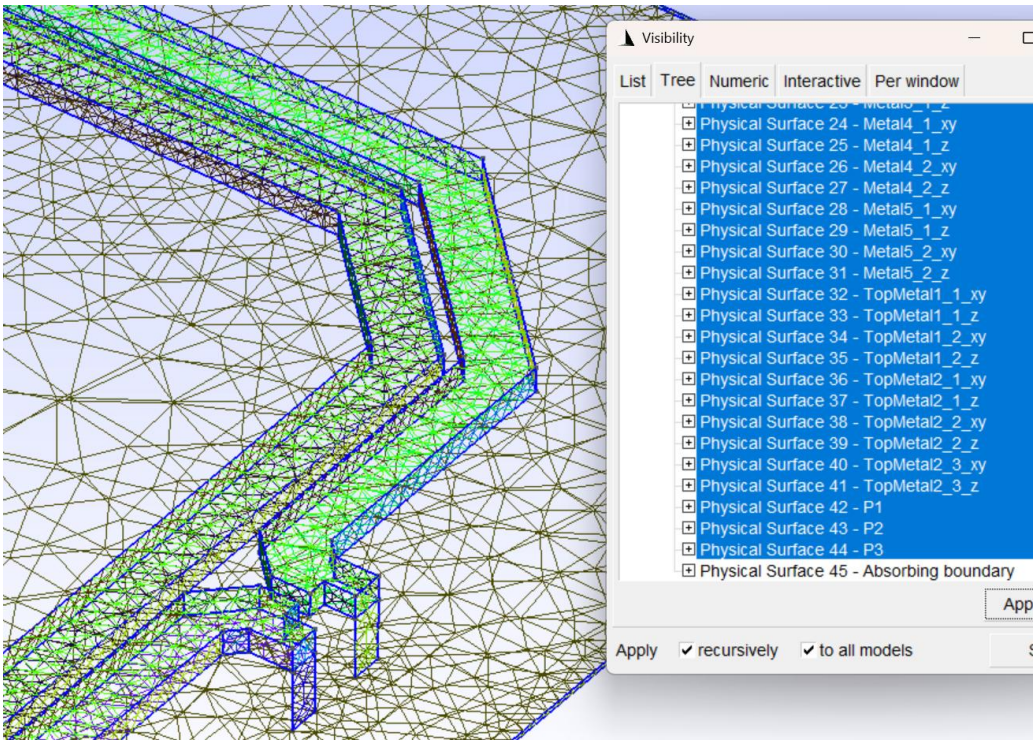
https://github.com/IHP-GmbH/TO_July2025/tree/main/FMD_QNC_D_Band_Phase_Shifter

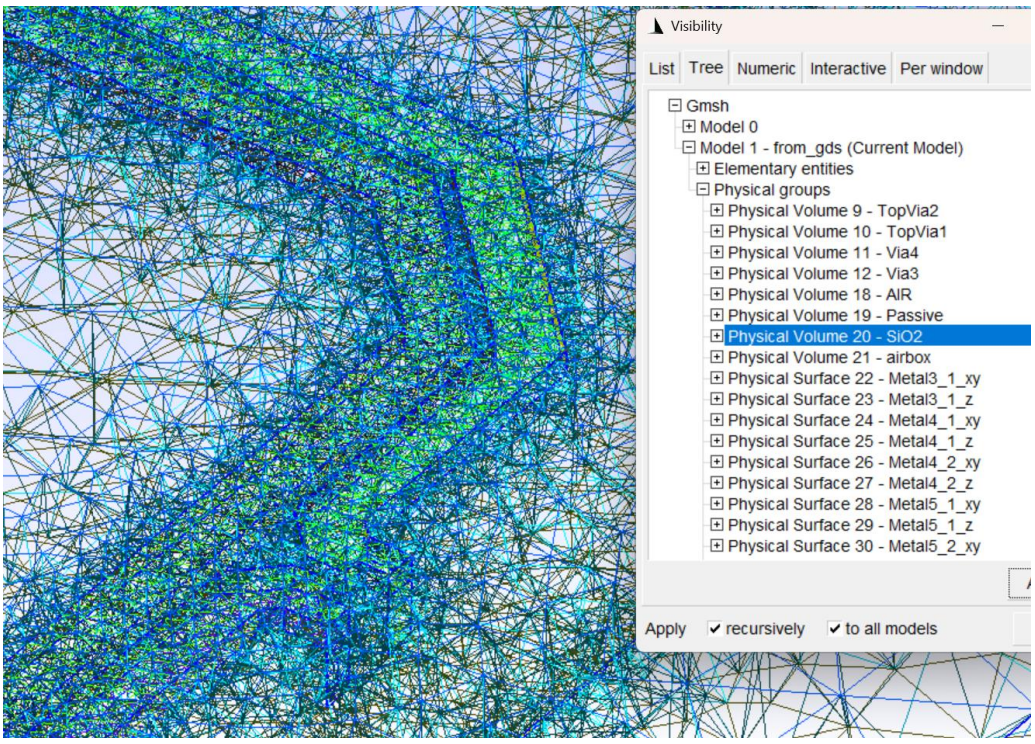


Line width is 7µm with TopMetal2 over Metal3 ground, gap width is 2µm.

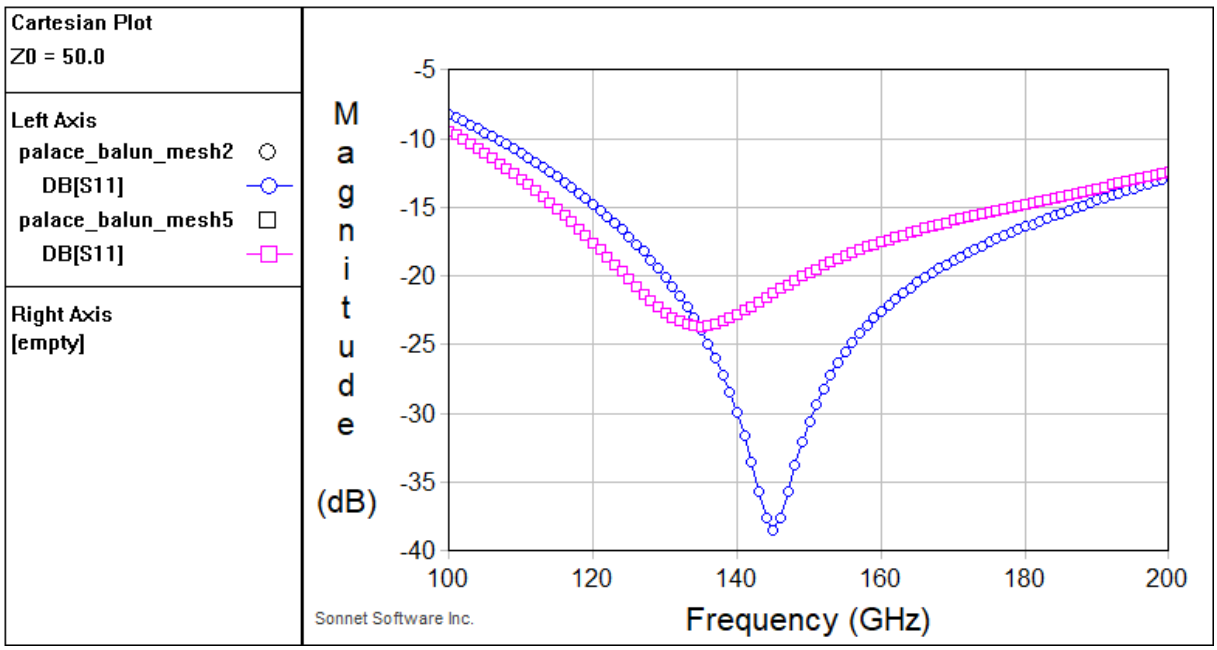


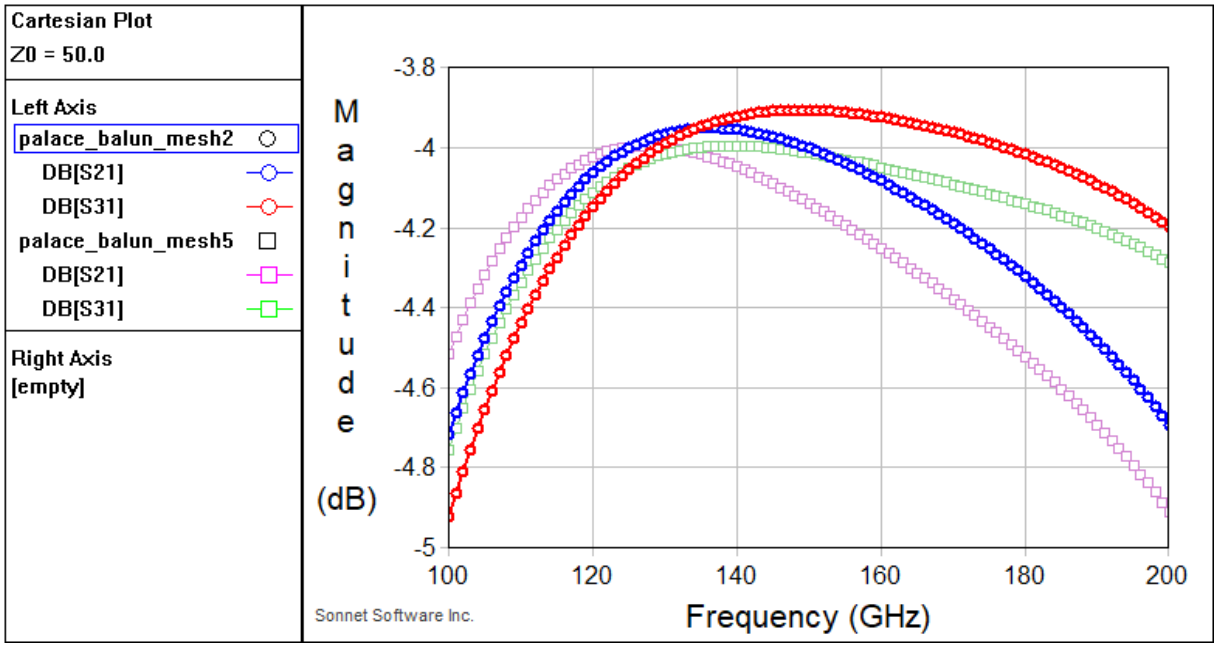
The critical dimension in this design is the 2µm wide gap between the coupler traces, the solver needs to model the field distribution in that gap with sufficient accuracy.





We will first look at results from refined_cellsize=5 and refined_cellsize=2, both with order=2, over the frequency band 100 to 200 GHz, with all 3 port excitations simulated for full S-parameter data.





There is quite a difference in S11 and S21,S31 results, and we can assume that the finer mesh gives more accurate results … but how far are we off from convergence?

We could now simulate with an ever smaller refined_meshsize = 1, or we can use adaptive mesh refinement now. This will run the simulation starting with an initial mesh size, and then refine the mesh in the most relevant regions.

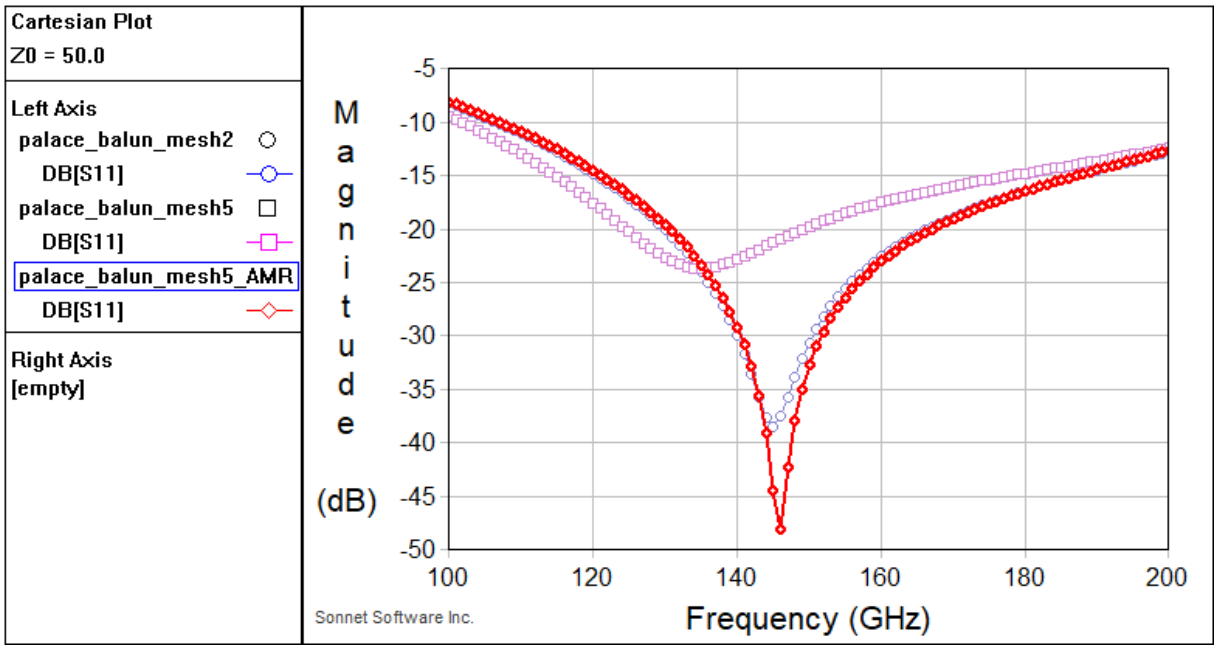
By default, adaptive mesh refinement in Palace uses data from all simulation frequencies and all port excitations, which means a significant increase in overall simulation time because a full simulation is used during mesh refinement.

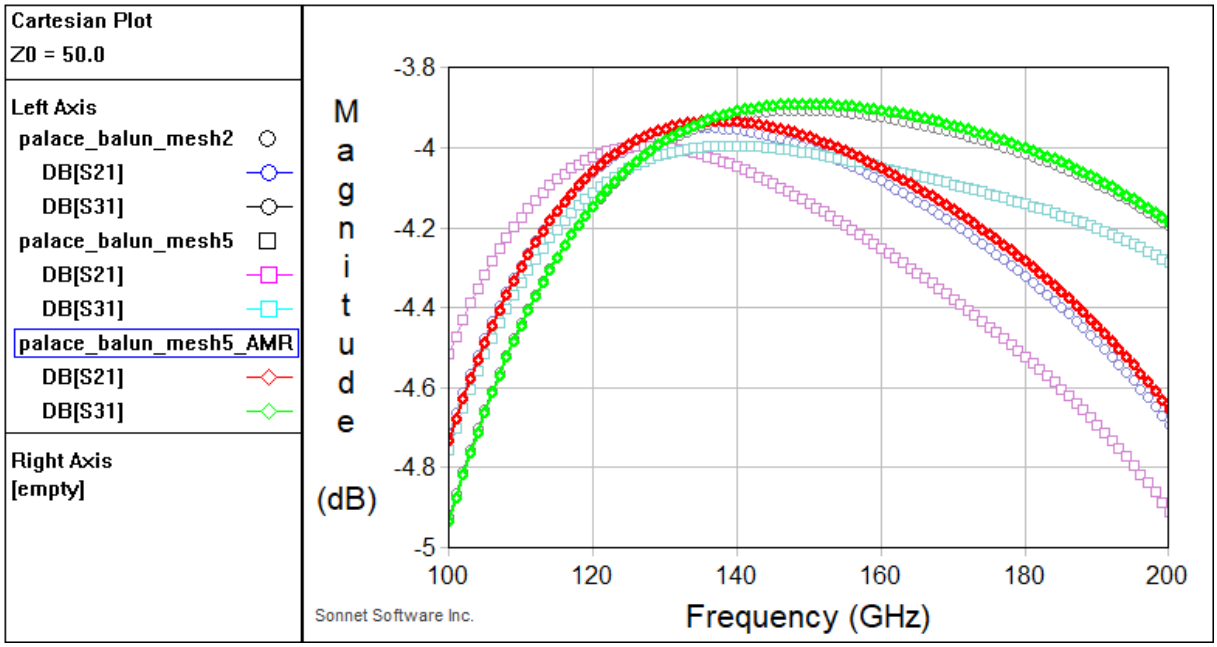
The screenshot below shows settings from the Palace configuration file for an adaptive sweep with maximum of 2 adaptive mesh refinements, with up to 2 million degrees of freedom.

```
    "Model": {
        "Mesh": "palace_balun_mesh5_AMR.msh",
        "L0": 1e-06,
        "Refinement": {
            "UniformLevels": 0,
            "Tol": 0.01,
            "MaxIts": 2,
            "MaxSize": 2000000.0,
            "Nonconformal": true,
            "UpdateFraction": 0.7,
            "SaveAdaptMesh": false
        }
    },
    "Solver": {
        "Driven": {
            "MinFreq": 100.0,
            "MaxFreq": 200.0,
            "FreqStep": 1.0,
            "SaveStep": 0,
            "AdaptiveTol": 0.01
        },
```

Note for experts: We could decide to create an adaptive mesh refinement at one target frequency, or a few selected target frequencies, and then store that refined mesh for the final full frequency sweep. This is the approach chosen by many commercial FEM solvers, where the user needs to decide what frequencies are used for AMR.
However, that is not implemented in this gds2gmsh workflow. The present implementation of adaptive mesh refinement in gds2gmsh workflow does each refinement step at all frequencies and for all port excitations, to provide a safe and reliable mesh refinement. In this case, the refined mesh exists in memory only and is not stored to disk.

Below is the result from an adaptive mesh refinement with 2 refinement steps, based on an initial mesh with refined_cellsize=5. This result is very close to the results obtained with a (fixed) mesh created from refined_cellsize=2, so both models come to similar results from different (initial) mesh.

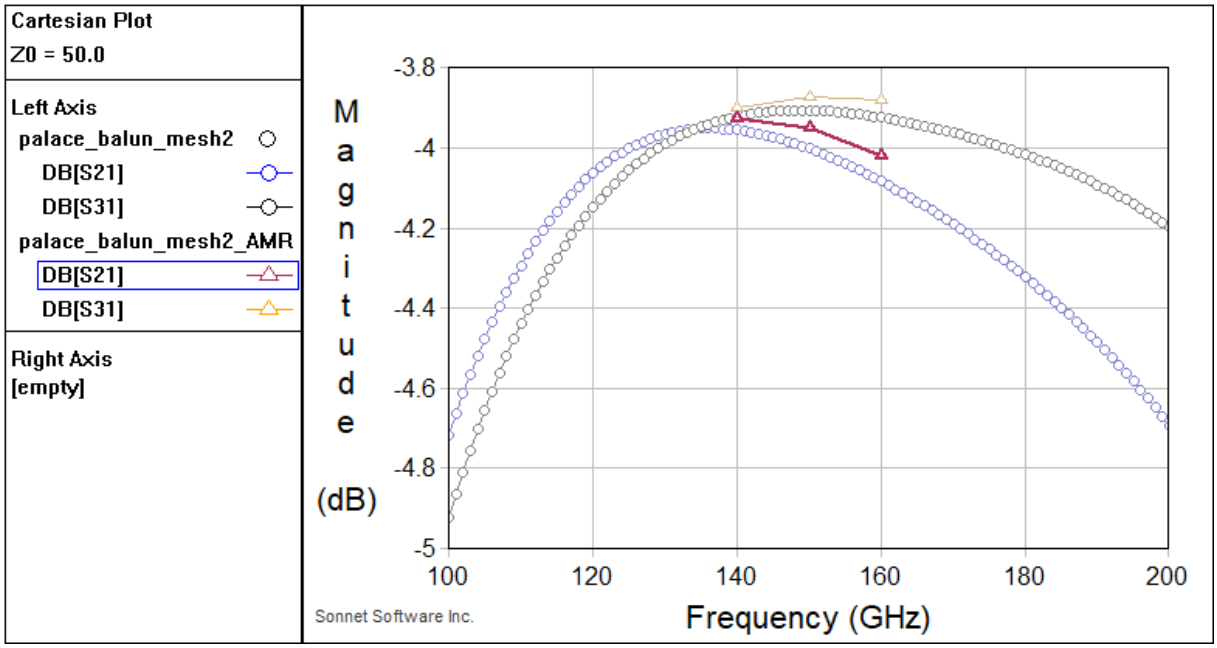


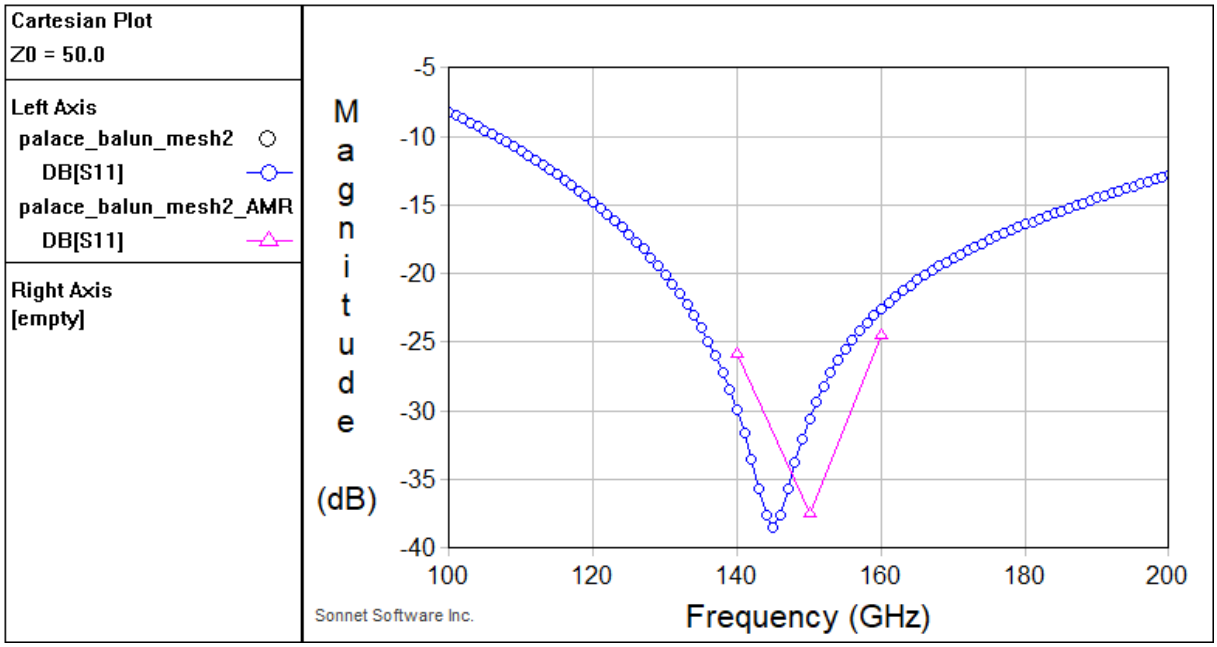


Here is the overview of simulation times and mesh size for full sweep over all 3 port excitations:

	Simulation time total	Degrees of freedom (from log)
refined_cellsize=2	580 s (13 freq)	836878
refined_cellsize=5	218 s (13 freq)	353056
refined_cellsize=5 + AMR 2x	1132 s (13 freq)	353056, 402138, 710236,

We could now spend more time to double check result with an adaptive mesh refinement starting from an initial mesh that is even finer, and to save time, that could be done at one or few frequencies only. Below is the result for 3 selected frequencies with 2 mesh refinements starting from 2µm:





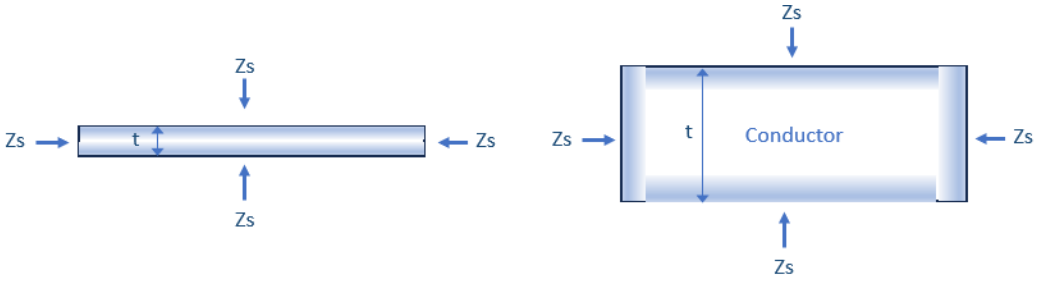
This gives an idea where a converged result would be located in frequency, at approximately 150 GHz resonance instead of 145 GHz calculated with coarser mesh.

Here is the overview of simulation times and mesh size for 3 frequencies (not full sweep!) over all 3 port excitations:

	Simulation time total	**Degrees of freedom (from log)**
refined_cellsize=2 + AMR 2x	1346 s (3 freq)	804166, 961866, 1742956

Conductor loss modelling

Many RF FEM solvers, including this gds2palace workflow, model metals as hollow bodies with surface resistance on the side walls. The advantage of this approach is that we don't need to mesh into skin effect, which might require sub-micron mesh size inside the conductors.



The surface impedance can be calculated easily when physical metal dimensions are much larger than skin depth. However, at low frequency and in the transition between skin effect regime and low frequency, we also need to consider conductor dimensions.

Limits of conductor loss calculation

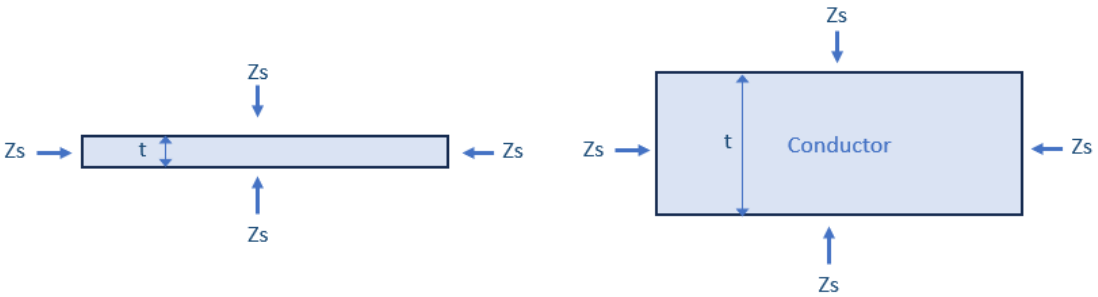
The gds2palace workflow creates surface elements for planar metals, and defines them as a conductor sheet with conductivity and thickness in the Palace config.json file. Thickness in this case is the value defined for that metal in the XML stackup file, e.g. 3µm thickness for TopMetal2.

The code in Palace does not know anything about the aspect ratio of the conductor, and uses a surface impedance calculation that is accurate for wide sheets, where we have a conductor that is much wider than the conductor height. Summing up the surface impedance mapped to top and bottom side of a thin sheet, the resulting impedance would be accurate down to DC where it is limited by physical conductor cross section.

However, in our RFIC use case, we might have conductor aspect ratio that reaches almost square shape, and we need to account for side wall currents, not only for closely spaced coupled lines.

Mapping the surface impedance (as calculated internally by Palace from conductor thickness) onto all surfaces, i.e. side walls as well as top and bottom, will lead to an over-estimate of available conductor cross section at low frequency:

The Palace calculation[3] of Zs is exact when applied to top and bottom side only. If we apply this value to all sides, the extreme case at DC will over-estimate the conductor cross section by factor (width+thickness)/width.



[3] https://github.com/awslabs/palace/blob/main/palace/models/surfaceconductivityoperator.cpp

The approach taken by the gds2palace workflow script is to have a **scaling factor** that **reduces** the thickness value for the side walls. This reduces the error at low frequency (where skin depth is not much smaller than physical dimensions) and we converge towards the precise result in skin effect regime.

In the model code, you can set this value by parameter 'z_thickness_factor':

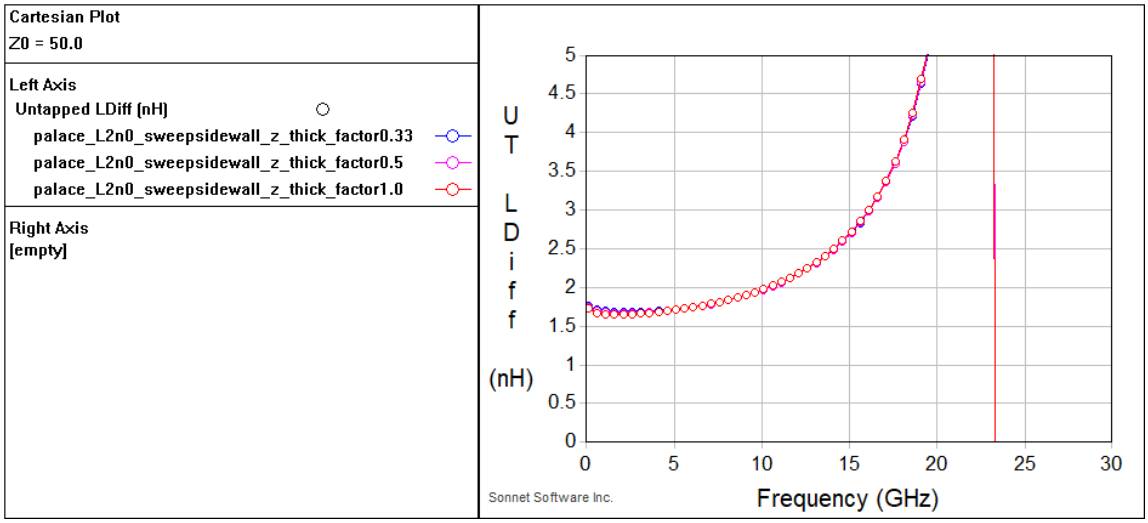
```
settings['meshsize_max'] = 30  # microns, override cells_per_wavelength
settings['adaptive_mesh_iterations'] = 0
settings['z_thickness_factor'] = 0.33
```

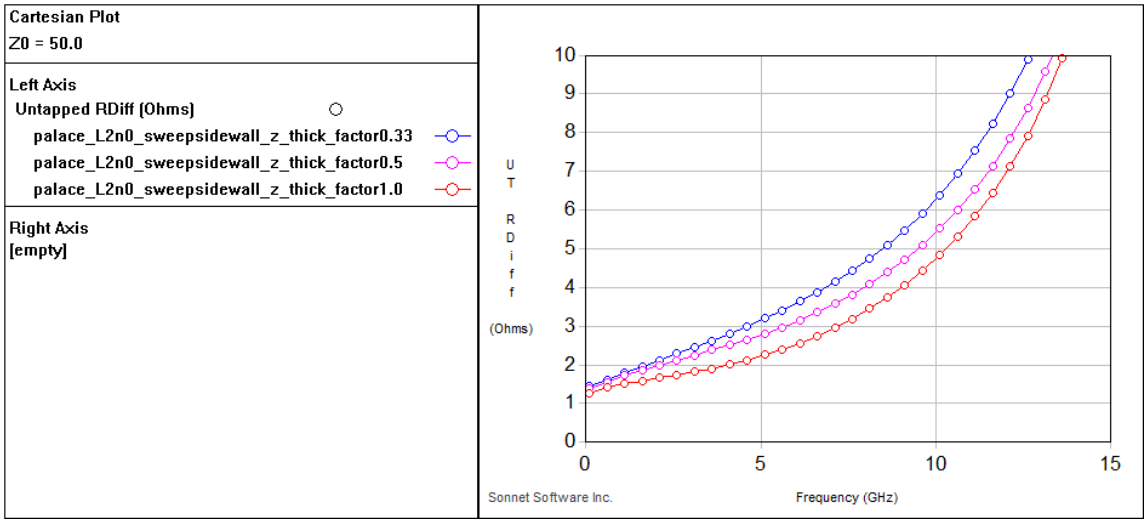
If you do not specify this option, the default value for that scaling factor is 1.0

Testcase L2n0 with z_thickness_factor 0.33, 0.5 and 1.0

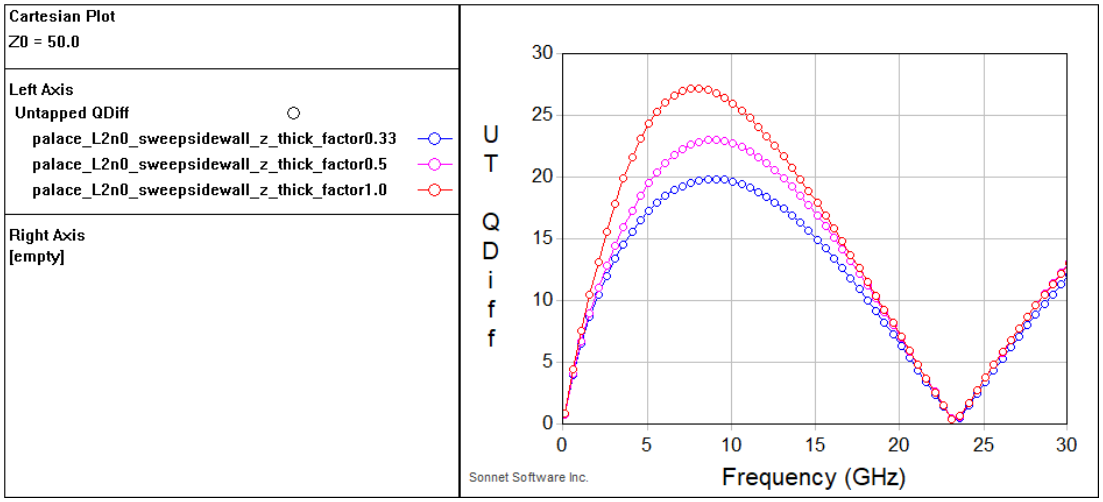
One testcase to check the effect of parameter z_thickness_factor is the 2nH inductor already known from IHP openEMS workflow.

The simulated inductance in differential model is visually identical for all 3 cases as expected, but differential resistance shows a significant difference.





Of course, this change in series resistance also shows up in Q factor:



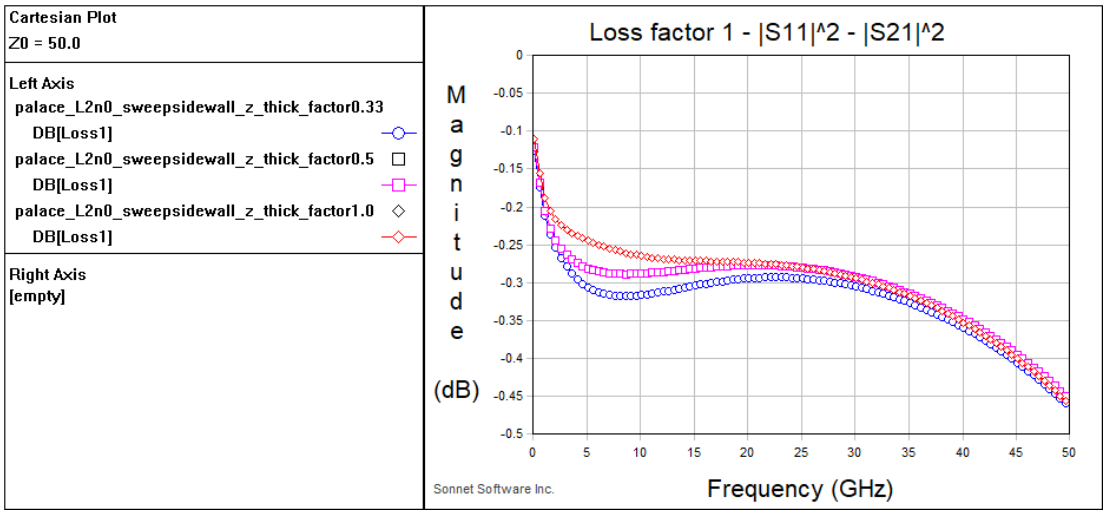
It was confirmed by single frequency simulations near peak Q that this change is not an artefact from adaptive frequency sweep.

This is not what we expected, because conductor width is 12µm. We do not have an extreme case where side walls give a massive over-estimate of total DC cross section, and indeed the low frequency resistance at 0.1 GHz is quite close: 1.27 Ohm @ z_thickness_factor = 0.33, 1.40 Ohm @ 0.5 and 1.45 Ohm @ 1.0

The percentage difference increases in the medium frequency range, just around frequency of maximum Q for this testcase.

To understand this, we check the skin depth of TopMetal2 in IHP SG13 technology: 1.29 µm @ 5 GHz. The skin effect correction done in internally in Palace calculates a correction factor from skin depth and half physical thickness, and at these frequencies both values are roughly the same, so that a correction of effective surface impedance takes place. By applying a correct factor on thickness used for side walls, we change surface impedance well into the skin effect regime!

To check losses more wideband, we now look at “loss factor” which is the relative amount of power dissipated in the simulation model. The plot below shows that the main difference between the different runs for this model is in the frequency range 2 to 10 GHz, and results converge at higher frequencies. The Q factor comparison for the L2n0 testcase falls into the range of worst error.



To verify that the difference is not caused by insufficient mesh density, adaptive mesh refinement was carried out with 3 iterations at 5 GHz: loss factor changed by no more than 0.05 dB with those 3 steps of mesh refinement.

The calculation below reproduces the surface impedance correction in Palace source code for two frequencies: 5 GHz (left columns) and 50 GHz (right column).

Palace code:

```
double delta = std::sqrt(2.0 / (bdr.mu * bdr.sigma * omega));
std::complex<double> Z = 1.0 / (bdr.sigma * delta);
Z.imag(Z.real());
if (bdr.h > 0.0)
{
  double nu = bdr.h / delta;
  double den = std::cosh(nu) - std::cos(nu);
  Z.real(Z.real() * (std::sinh(nu) + std::sin(nu)) / den);
  Z.imag(Z.imag() * (std::sinh(nu) - std::sin(nu)) / den);
}
```

Simple DC cross section calculation		
conductivity S/m	3,03E+07	3,03E+07
thickness µm	3,0	3,0
Rs, DC from geometry 1/(sigma*t)	**1,10E-02**	**1,10E-02**

Source code:		
https://github.com/awslabs/palace/blob/main/palace/models/surfaceconductivityoperator.cpp		
Calculation steps per source code:		
freq	5,0E+09	5,0E+10
omega	3,14E+10	3,14E+11
µ0	1,26E-06	1,00E+00
delta (skin depth)	1,29E-06	4,09E-07
nu	2,32E+00	7,33E+00
den	5,82E+00	7,66E+02
Zs_re from skin effect (no modif.)	2,55E-02	8,07E-02
Zs_im from skin effect (no modif.)	2,55E-02	8,07E-02
factor_re	0,99	1,00
factor_im	0,74	1,00
Zs_re including modif. factor	**2,53E-02**	**8,08E-02**
Zs_im including modif. factor	1,89E-02	8,07E-02

At 5 GHz, skin depth is on the order of half conductor thickness (TopMetal2 thickness is 3µm) and the correction of real and imaginary part of Zs takes effect. The correction factor used at 5 GHz for Re{Zs} is 0.99 whereas the correction factor used for Im{Zs} is 0.74

If we now apply a different thickness to the side walls, to achieve a correction of available cross section at low frequencies, this has a strong effect on correction values calculated by Palace.

The previous table showed that on the top/bottom side where metal thickness is specified as 3µm, the correction factors are 1.0 at 50 GHz.

If we use z_thickness_factor = 0.33, so that side wall sheets are specified as 3µm * 0.33 = 1µm thickness, this results in massive changes at 5 GHz but even the 50 GHz imaginary part of Zs on the side walls is changed significantly.

Simple DC cross section calculation		
conductivity S/m	3,03E+07	3,03E+07
thickness µm	1,0	1,0
Rs, DC from geometry 1/(sigma*t)	**3,30E-02**	**3,30E-02**

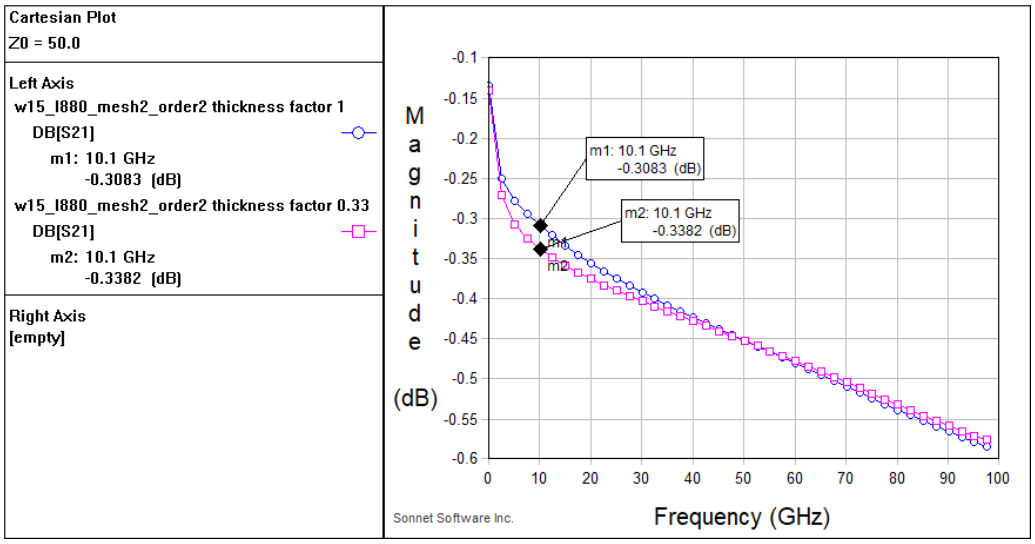
Source code:		
https://github.com/awslabs/palace/blob/main/palace/models/surfaceconductivityoperator.cpp		
Calculation steps per source code:		
freq	5,0E+09	5,0E+10
omega	3,14E+10	3,14E+11
µ0	1,26E-06	1,00E+00
delta (skin depth)	1,29E-06	4,09E-07
nu	7,73E-01	2,44E+00
den	5,98E-01	6,58E+00
Zs_re from skin effect (no modif.)	2,55E-02	8,07E-02
Zs_im from skin effect (no modif.)	2,55E-02	8,07E-02
factor_re	2,59	0,97
factor_im	0,26	0,77
Zs_re including modif. factor	**6,61E-02**	**7,81E-02**
Zs_im including modif. factor	6,57E-03	6,23E-02

In other word, the we do not only change low frequency losses, but we also change high frequency impedance on the side walls, especially the imaginary part.

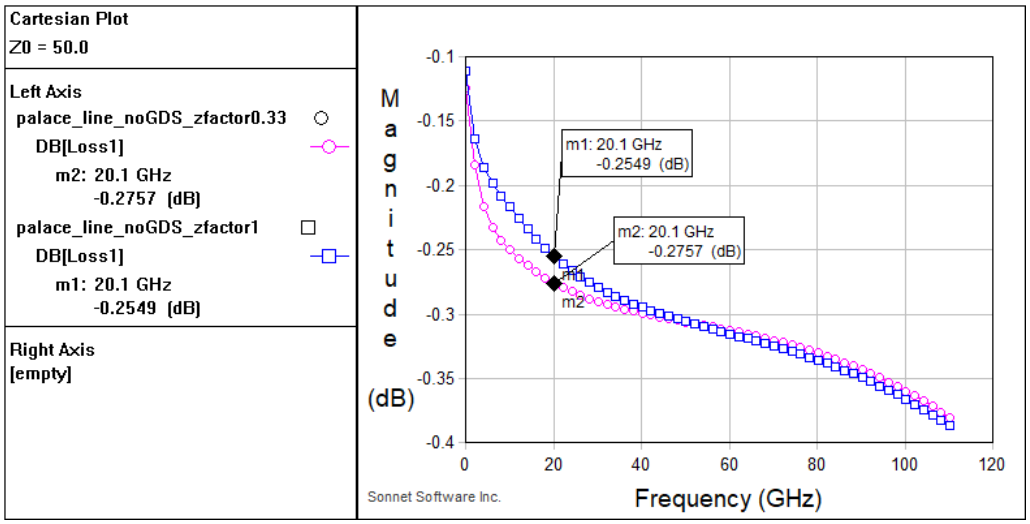
Testcase microstrip line

As another test case, we use the 50 Ohm microstrip line from a previous chapter, where the mesh density and basis functions were investigated. The line is 15µm wide on TopMetal2 over Metal1 ground, with 880µm line length.

Below is a plot of S21 where we can see some difference between z_thickness_factor=0.33 and z_thickness_factor=1.0 in insertion loss, especially in the frequency range up to 40 GHz.



When testing a narrower line with 7µm TopMetal2 over Metal3, this is not matched to 50 Ohm very well and S21 would show some ripple, so we better look at loss factor 1 - $|S11|^2$ - $|S21|^2$ to compare the metal losses between z_thickness_factor=0.33 and z_thickness_factor=1.0



This testcases uses a shorter line length of 500µm only, for faster simulation, so the total insertion loss of the shorter line is lower. Both 15µm line and 7µm line testcase were simulated with refined_cellsize = 2 and mesh order = 2, with no adaptive mesh refinement.

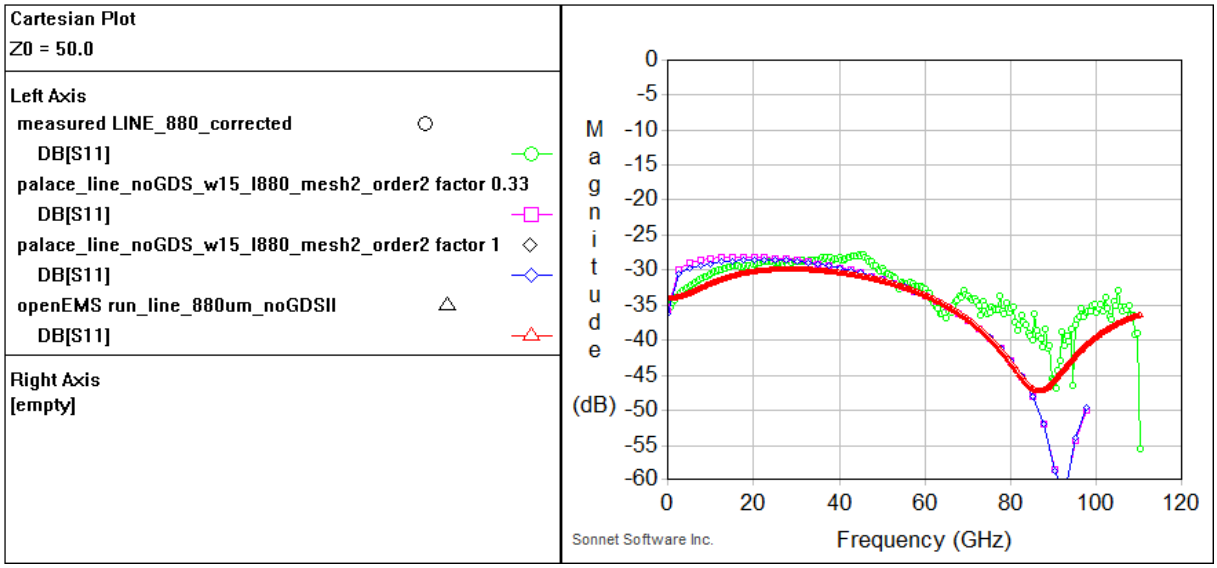
Compare to other data

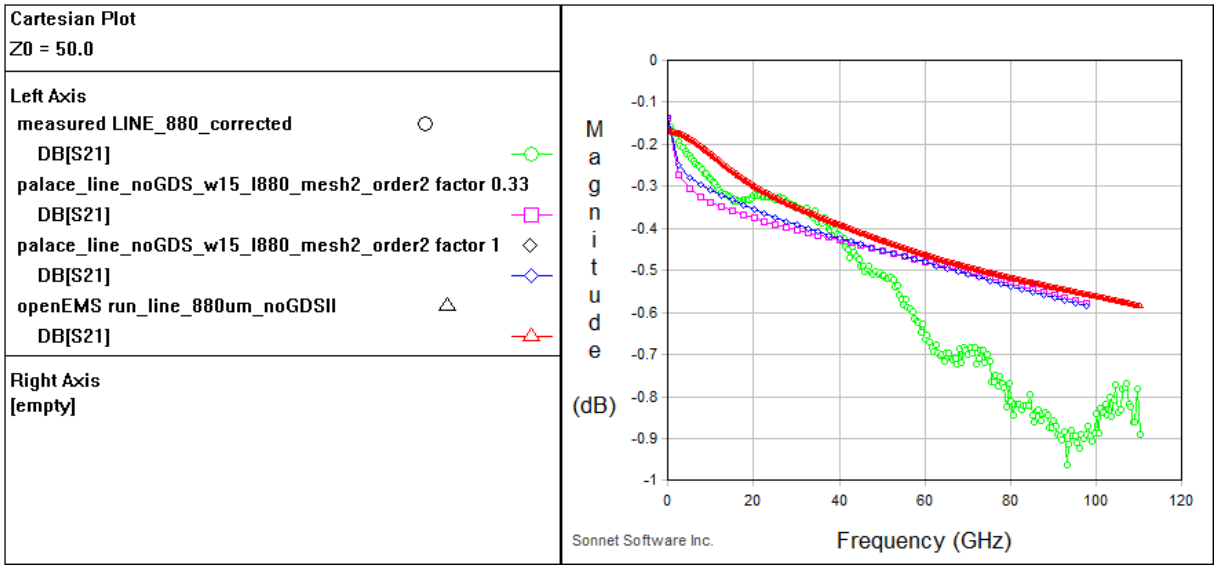
We can also compare simulated results of the line with 15µm TopMetal2 over Metal1 to other data sources: openEMS simulation and measurement.

Palace simulations for this line were done without the substrate below, the stackup didn't include the substrate because we expect the Metal1 ground plane to shield the substrate from the fields in the transmission line.[4]

For openEMS (red curve), the simulation was also done without substrate below, using metal (PEC) side walls. Mesh refinement was set to 0.5µm because openEMS needs to mesh into skin effect.

The measurement (light blue) was obtained by de-embedding the line from the overall measurement data with GSG pads, so the overall ripple might be partially due to this more complex setup.

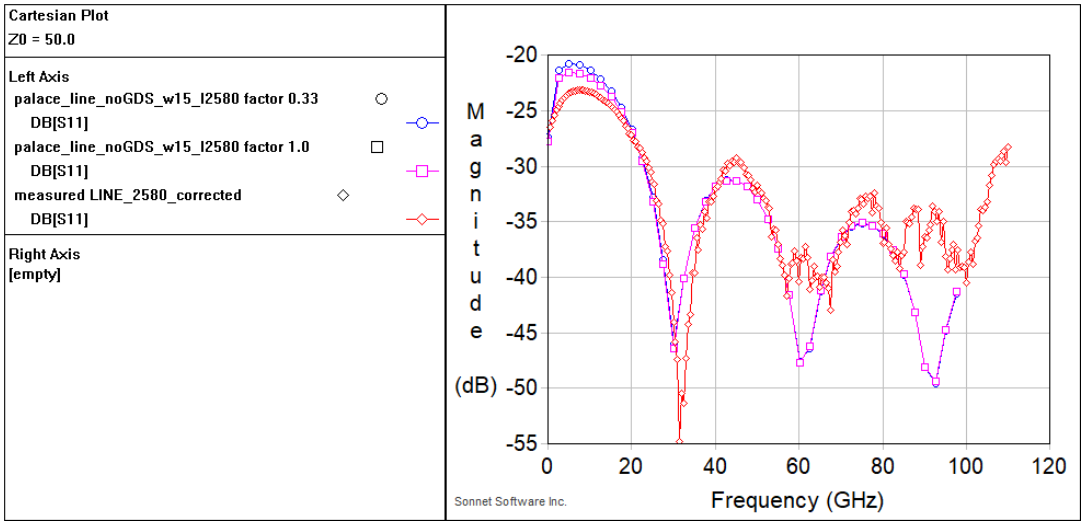


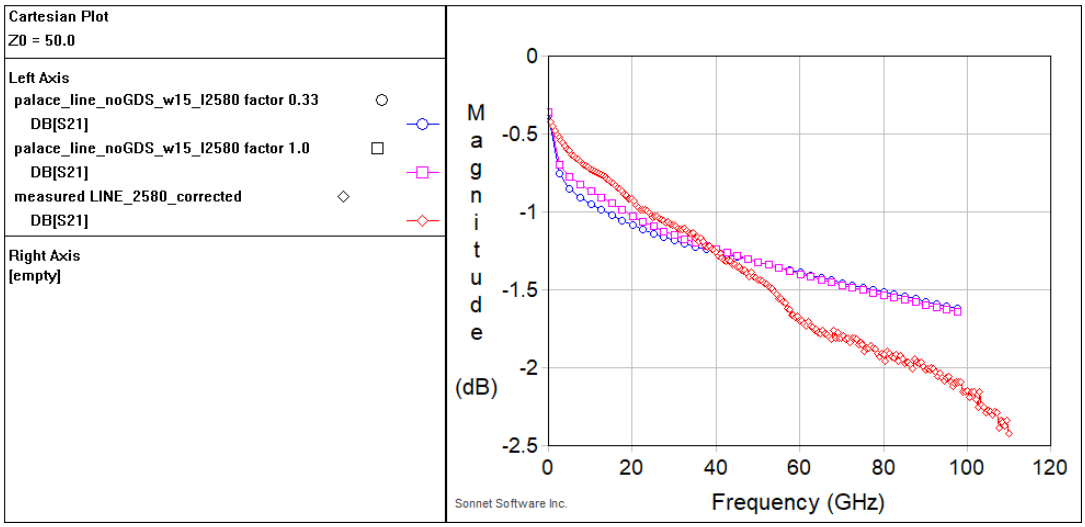


Looking at the S21 plot, the blue curve (Palace with z_thickness_factor=1) is closer to the others than the pink curve with z_thickness_factor=0.33

[4] It was verified that the difference with/without substrate below is negligible for these line models.

We can also compare data for a longer line of 2580µm, width 15µm TopMetal2 over Metal1.





The pink curve (Palace with z_thickness_factor=1) is closer to measured data up to 40 GHz. Above that frequency range, z_thickness_factor has very little effect.

It can also be seen that measured insertion increases more with frequency than simulated. One possible explanation from an earlier paper was dielectric loss in the SiO2. This dielectric was modelled here as lossless dielectric, loss tangent = 0.
If you prefer to use another value for loss tangent, that is fully supported by the workflow, and can be specified easily in the XML stackup file.

Conclusion regarding z_thickness_factor

Reduced thickness for the side walls looked like a simple, effective correction to prevent an over-estimate of low frequency conductor cross section, but this effect spreads out into the microwave frequency range. Further investigation is needed on this topic, to see which default setting is most appropriate.

Dielectric loss modelling

The transmission line examples in the metal loss chapter showed a difference between simulated and measured insertion loss which increases toward higher frequencies.

As already mentioned in that metal loss chapter, this could indicate higher **dielectric loss** than modelled. In the supplied XML stackup files, SiO2 is modelled as a lossless dielectric. To verify if dielectric loss could be the reason for poor agreement in that frequency range, we repeat the simulation of the 2580 µm line with non-zero loss tangent for the SiO2 layer.

Loss tangent value used in this simulation is **0.01**, which is a rough estimate based on a paper published by IHP authors for SG25H technologies back in 2007:

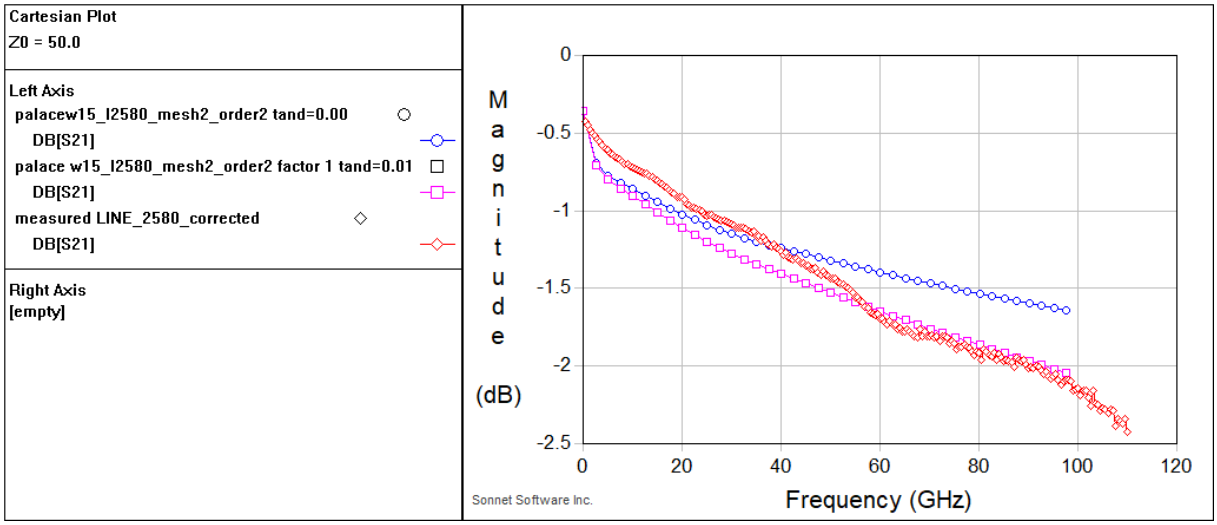
Korndörfer F, F Sischka Optimization of the Substrate Parameters for EM-simulators MOS-AK/ESSDERC/ESSCIRC Workshop (Munich) 14 Sep., 2007

https://www.mos-ak.org/munich_2007/posters/P05_MOS-AK_Korndoerfer.pdf

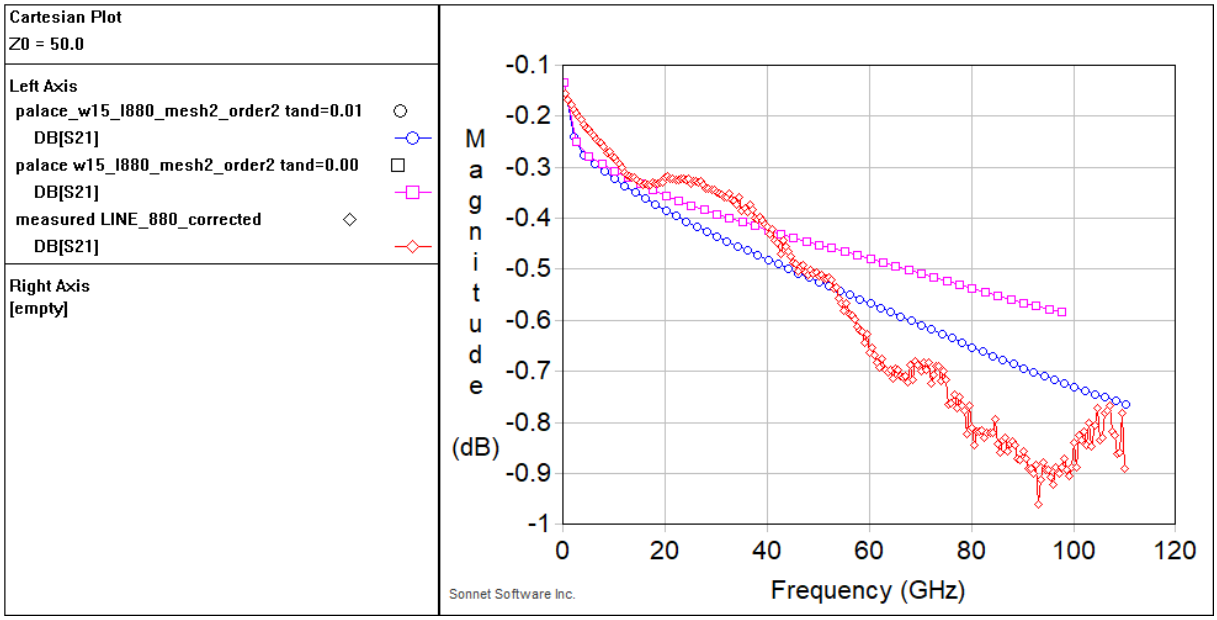
TABLE II
SUMMARY OF TEST STRUCTURES AND RESULTS

Test structure	Area in µm²	Measured capacitance @1 MHz in fF	Measured capacitance @10 GHz in fF	Start values		Optimized values	
				ε_r	Loss tangent	ε_r	Loss tangent
M1-M2	5307	270	265	4.1	0	4.27	0.009
M2-M3	5307	250	245	4.1	0	4.37	0.006
M3-M4	5307	245	235	4.1	0	4.14	0.008
M4-M5	5307	85	80	4.1	0	4.28	0.014

Indeed, that simulation with SiO2 loss tangent of 0.01 (pink curve) gives increased high frequency loss and quite nice agreement with the measured data above 50 GHz, much better than the model with lossless SiO2.



For the 880µm line length, the ripple on measurement data has a larger (relative) impact, but we see the same trend: simulation results with tand=0.01 are much closer to measurement than the simulation without SiO2 losses.



Conclusion regarding dielectric losses

The existing data is not complete enough to create an “official” technology stackup with SiO2 losses, but it seems that some amount of dielectric loss is real.

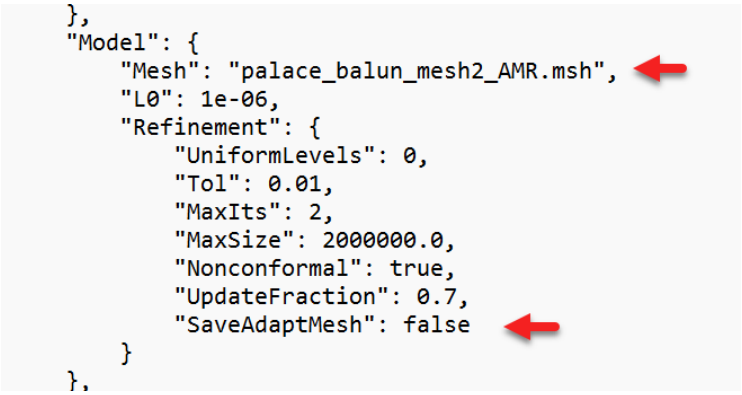
If you decide to include dielectric loss in your simulation, this can be easily specified in the XML stackup file using parameter *DielectricLossTangent* for the SiO2 material definition.

Advanced topics

Adaptive mesh refinement at selected frequencies only

By default, adaptive mesh refinement in Palace uses data from all simulation frequencies and all port excitations, which means a significant increase in overall simulation time because a full simulation is used during mesh refinement.

To do a two-step approach where the adaptive mesh refinement used only a limited set of frequencies, you could run the gds2palace workflow with those frequencies only, and change the config.json file to **store the resulting mesh data to disk**, by setting SaveAdaptMesh to true.



Adaptive frequency sweep would be **disabled** by removing the "AdaptiveTol" line in the frequency block.

After running Palace with this config file, you will find a refined mesh file in the output folder. Note that the final mesh refinement result is **not** located in the “iteration_nn” folder, the results of adaptive sweep can be found one level above (!) in the parent folder.

You would then create another config.json file with the full frequency range and adaptive frequency sweep enabled, with the Mesh setting pointing to that adaptive mesh file created by the AMR before.

Using wave ports instead of lumped ports

This gds2palace workflow creates lumped ports, which introduce some physical size and thus some parasitic series impedance.

The Palace solver also supports using wave ports, but that is not implemented in the gds2gmsh workflow. You could try to create the required mesh elements (sheet?) by faking a lumped port in the desired location, and then modify the port configuration in the palace.json simulation control file. Details on all Palace settings and options is available at https://awslabs.github.io/palace/stable/

Using S-Parameter output, model extraction

S-Parameters are an accurate wideband description of the simulation results, but can be difficult to use in time domain circuits simulators. In this chapter, we will look at some possible solutions.

Lumped circuit model extraction

For a limited range of devices, we can use straightforward calculation of the equivalent lumped circuit model. Such a calculation is available here, based on simple narrowband calculation:

https://github.com/VolkerMuehlhaus/lumpedmodel

Currently, these devices are supported:

- Untapped inductors (2-port)
- MIM capacitor (2-port)
- Transmission line model (RLGC, 2-port data)

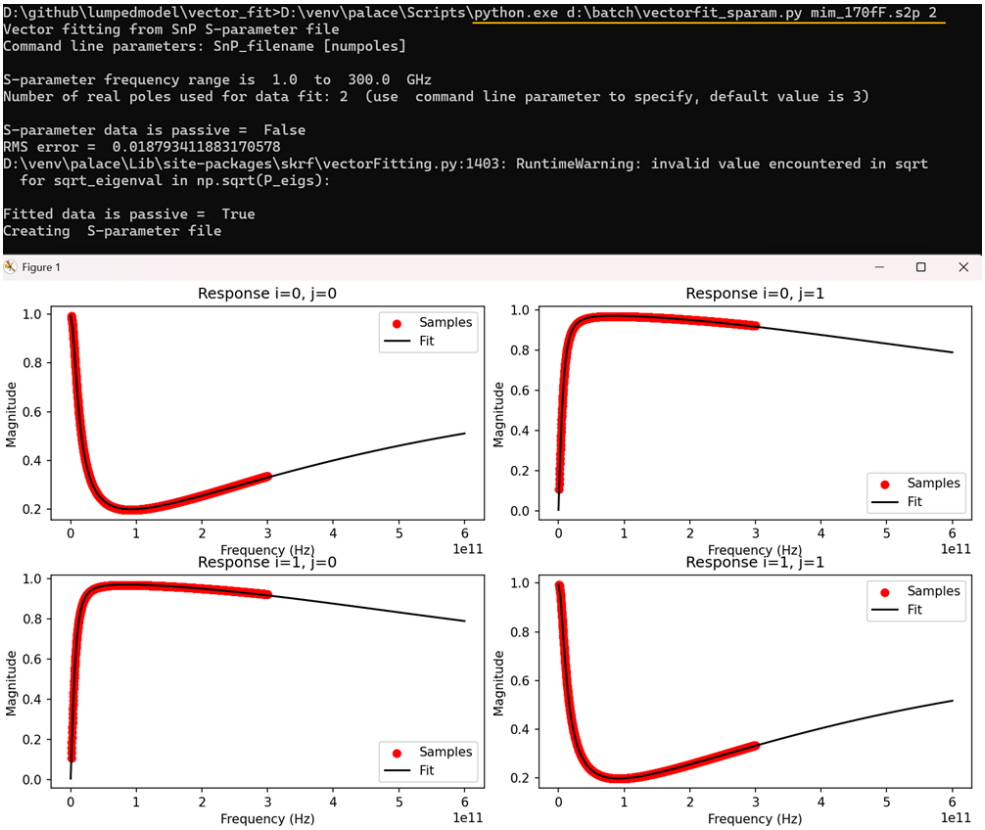
These tools are provided as Python scripts. The user needs to specify a target frequency, the fit result is then plotted along with the input data to show the quality of the model fit.

For sure, there are other extraction tools available elsewhere that might cover more use cases.

Mathematical “black box” vector fit

Another approach is to do vector fitting of arbitrary n-port data. One possible implementation is available here, using the vector fit in scikit-rf library:

https://github.com/VolkerMuehlhaus/lumpedmodel/tree/main/vector_fit



Appendix

Understanding volumes and surfaces created from GDSII

The workflow creates different types of geometry: volumes and surfaces. This depends on the mapping in the XML stackup file.

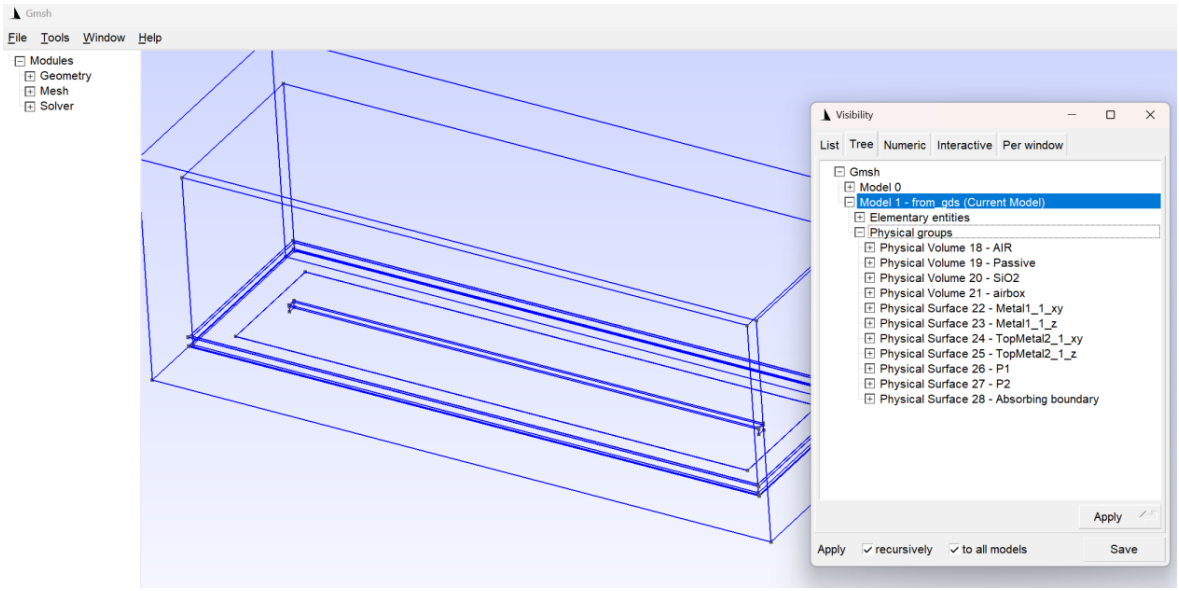
Geometries from GDSII layers are defined in the “Layers” section of the XML file :

- Type=”conductor” is used for all regular metal layers, in the mesh they are represented by **“Surface”** objects on all boundaries (top, bottom, side walls). In other words, these metals are represented by hollow shells and not by solid volumes.
- Type=”sheet” is used for metal layers that have no physical height in the model, they are represented as one flat **“Surface**” only.
- Type=”via” is used for via layers, these are represented in the mesh by **“Volume”** objects.

Extract from an XML stackup file, showing layer mappings with conductor, via and sheet types:

```
    </Dielectrics>
  ▼<Layers>
     <Substrate Offset="183.75"/>
     <Layer Name="Activ" Type="conductor" Zmin="0.0000" Zmax="0.4000" Material="Activ" Layer="1"/>
     <Layer Name="Metal1" Type="conductor" Zmin="1.0400" Zmax="1.4600" Material="Metal1" Layer="8"/>
     <Layer Name="Metal2" Type="conductor" Zmin="2.0000" Zmax="2.4900" Material="Metal2" Layer="10"/>
     <Layer Name="Metal3" Type="conductor" Zmin="3.0300" Zmax="3.5200" Material="Metal3" Layer="30"/>
     <Layer Name="Metal4" Type="conductor" Zmin="4.0600" Zmax="4.5500" Material="Metal4" Layer="50"/>
     <Layer Name="Metal5" Type="conductor" Zmin="5.0900" Zmax="5.5800" Material="Metal5" Layer="67"/>
     <Layer Name="TopMetal1" Type="conductor" Zmin="6.4303" Zmax="8.4303" Material="TopMetal1" Layer="126"/>
     <Layer Name="TopMetal2" Type="conductor" Zmin="11.2303" Zmax="14.2303" Material="TopMetal2" Layer="134"/>
     <Layer Name="TopVia2" Type="via" Zmin="8.4303" Zmax="11.2303" Material="TopVia2" Layer="133"/>
     <Layer Name="TopVia1" Type="via" Zmin="5.5800" Zmax="6.4303" Material="TopVia1" Layer="125"/>
     <Layer Name="Via4" Type="via" Zmin="4.5500" Zmax="5.0900" Material="Via4" Layer="66"/>
     <Layer Name="Via3" Type="via" Zmin="3.5200" Zmax="4.0600" Material="Via3" Layer="49"/>
     <Layer Name="Via2" Type="via" Zmin="2.4900" Zmax="3.0300" Material="Via2" Layer="29"/>
     <Layer Name="Via1" Type="via" Zmin="1.4600" Zmax="2.0000" Material="Via1" Layer="19"/>
     <Layer Name="Cont" Type="via" Zmin="0.4000" Zmax="1.0400" Material="Cont" Layer="6"/>
     <Layer Name="SUBGND" Type="conductor" Zmin="-3.75" Zmax="0" Material="LOWLOSS" Layer="210"/>
     <Layer Name="BACKSIDEGND" Type="sheet" Zmin="-180" Zmax="-180" Material="LOWLOSS" Layer="211"/>
     <Layer Name="MIM" Type="conductor" Zmin="5.6043" Zmax="5.7540" Material="MIM" Layer="36"/>
     <Layer Name="Vmim" Type="via" Zmin="5.7540" Zmax="6.4303" Material="Vmim" Layer="129"/>
     <Layer Name="LBE" Type="dielectric" Zmin="-183.75" Zmax="0" Material="Air" Layer="157"/>
   </Layers>
  </ELayers>
</Stackup>
```

Resulting mesh with surface and volume objects:



Dielectrics: There are additional materials in the stackup that are not drawn in the GDSII file, such as substrate or oxide or passivation. These are created as **"Volume"** objects in the mesh.

Dielectrics are defined in the <Dielectrics> section of the XML file, and usually cover the entire drawing area (bounding box of GDSII drawing) plus an additional "margin" that is defined in the simulation model code.

```
  <Material Name="MIM" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="500000.0
 </Materials>
▼<ELayers LengthUnit="um">
 ▼<Dielectrics>
    <Dielectric Name="AIR" Material="AIR" Thickness="200.0000"/>
    <Dielectric Name="Passive" Material="Passive" Thickness="0.4000"/>
    <Dielectric Name="SiO2" Material="SiO2" Thickness="15.7303"/>
    <Dielectric Name="EPI" Material="EPI" Thickness="3.7500"/>
    <Dielectric Name="Substrate" Material="Substrate" Thickness="180.0000"/>
  </Dielectrics>
 ▼<Layers>
    <Substrate Offset="183.75"/>
    <Layer Name="Activ" Type="conductor" Zmin="0.0000" Zmax="0.4000" Material="Activ" Layer="1"/>
```

As an option, the size of these dielectric layers can be defined by an optional parameter **Boundary** which sets the size of the dielectric from the bounding box of that layer in the GDSII file.

```
  <Material Name="Underfill" Type="Dielectric" Permittivity="1.0" DielectricLossTangent="0.0" Conducti
 </Materials>
▼<ELayers LengthUnit="um">
 ▼<Dielectrics>
    <Dielectric Name="iAIR" Material="AIR" Thickness="50.0000" Boundary="1039"/>
    <Dielectric Name="iSubstrate" Material="Substrate" Thickness="80.0000" Boundary="1039"/>
    <Dielectric Name="iSiO2" Material="SiO2" Thickness="12.27" Boundary="1039"/>
    <Dielectric Name="iPassive" Material="Passive" Thickness="0.4000" Boundary="1039"/>
    <Dielectric Name="Underfill" Material="Underfill" Thickness="46.2" Boundary="1039"/>
    <Dielectric Name="Passive" Material="Passive" Thickness="0.4000" Boundary="39"/>
    <Dielectric Name="SiO2" Material="SiO2" Thickness="15.7303" Boundary="39"/>
    <Dielectric Name="EPI" Material="EPI" Thickness="3.7500" Boundary="39"/>
    <Dielectric Name="Substrate" Material="Substrate" Thickness="180.0000" Boundary="39"/>
    <Dielectric Name="AIR" Material="AIR" Thickness="50.0000" Boundary="39"/>
  </Dielectrics>
 ▼<Layers>
    <Substrate Offset="233.75"/>
    <Layer Name="Activ" Type="conductor" Zmin="0.0000" Zmax="0.4000" Material="Activ" Layer="1"/>
```

Mapping of Volumes and Surfaces to Palace materials

Palace offers a wide range of options to assign material properties to volumes and surfaces.
Our workflow creates that mapping in the Palace control file (config.json), as described below:

Via layers (Type="via") are created as volumes, and the conductivity defined in the XML stackup is assigned to these volumes.

```
  "Domains": {
      "Materials": [
          {
              "Attributes": [
                  9                    Physical volume #
              ],
              "Permittivity": 1.0,
              "Conductivity": [
                  314300.0,
                  314300.0,
                  3143000.0
              ]
          },
```

To account for the z-directed nature of via arrays, which allows current flow predominantly in z direction, the conductivity from XML is only assigned to z-direction, and the value in xy-direction is reduced by a factor of 10. This avoids issues with "unreal" currents flowing on the side walls of merged via polygons after via array merging.

Metal layers (Type="conductor") are created as hollow elements surrounded by surfaces, with surface impedance to define metal loss. The conductivity and thickness are obtained from the XML file.

```
"Boundaries": {
    "Conductivity": [
        {
            "Attributes": [
                27                   Physical volume #
            ],
            "Conductivity": 21640000.0,
            "Thickness": 0.41999999999999993
        },
```

This is straightforward for top and bottom of the conductors, with the total conductor cross section being calculated properly even at low frequency where skin depth is larger than the physical conductor height.

However, using that same surface impedance also for the side walls, we would over-estimate the total physical cross section, so the side walls get a different material mapping with a thickness value that is only 1/3 of the top/bottom values. This gives a reasonable result also in the transition from skin effect regime to low frequency.

Metall sheet layers (Type=”sheet”) are created as a single, flat surface. Their position in the stackup should be defined with Zmin=Zmax for better readability, but actually the Zmax value is ignored here.

```
<Layer Name="MIM" Type="conductor" Zmin="5.6043" Zmax="5.7540" Material="MIM" Layer="36"/>
<Layer Name="Vmim" Type="via" Zmin="5.7540" Zmax="6.4303" Material="Vmim" Layer="129"/>
<Layer Name="LBE" Type="dielectric" Zmin="-183.75" Zmax="0" Material="Brick" Layer="157"/>
<Layer Name="ResistorA" Type="sheet" Zmin="22.0" Zmax="22.0" Material="ResA" Layer="220"/>
<Layer Name="ResistorB" Type="sheet" Zmin="15.0" Zmax="15.0" Material="ResB" Layer="221"/>
```

These sheets are typically used with a Type=”Resistor” material definition in the XML file, with fixed sheet resistance in Ohm/square.

```
<Material Name="Vmim" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="2191000.0" Color="ffe6bf"/>
<Material Name="MIM" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="500000.0" Color="e6ffbf"/>
<Material Name="LBE" Type="Dielectric" Permittivity="1.0" DielectricLossTangent="0.0" Conductivity="0" Color="d0d0d0"/>
<Material Name="ResA" Type="Resistor" Rs="3.5e-1" Color="d0d0d0"/>
<Material Name="ResB" Type="Resistor" Rs="1.1" Color="d0d0d0"/>
```

In the Palace config file, this results in an Impedance boundary with Rs value mapped to the surface.

```
"Impedance": [
    {
        "Attributes": [
            37,
            38
        ],
        "Rs": 0.35
    },
    {
        "Attributes": [
            39
        ],
        "Rs": 1.1
    }
],
```

Physical volume #

Physical volume #

Software versions used in this document

Palace version used for this document: v0.14

Palace installation method used: container for Apptainer environment, running on Ubuntu 24.04.02 LTS

Simulation host: AMD Ryzen 9-7950X (16 cores) with 128 GB RAM

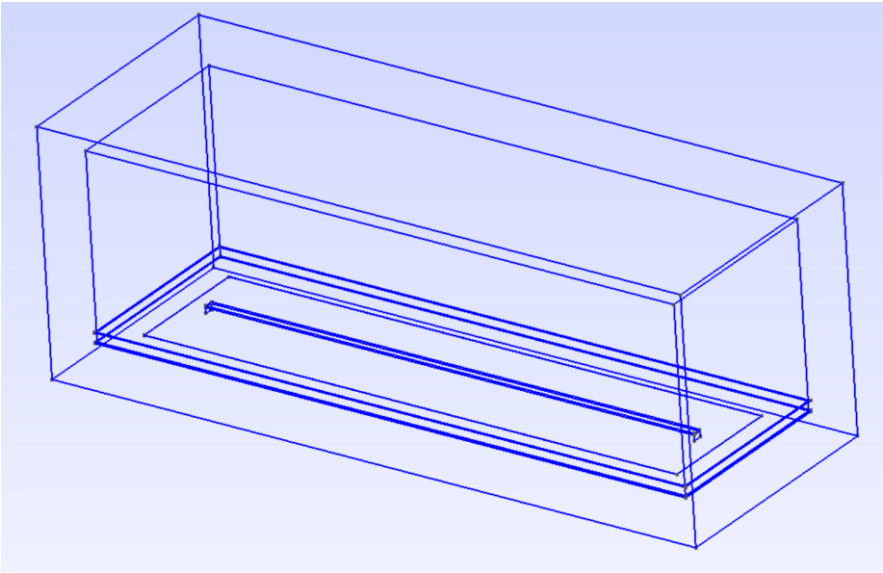
Code base used for this test:
gds2palace Python scripts as of October 13, 2025 (unless noted otherwise)

List of examples

Here is an overview of model examples.

palace_line_viaport.py

Microstrip line on TopMetal2 over Metal1 ground plane, with via ports on both ends.
Geometry is read from GDSII. The stackup does not include bulk silicon because that is shield by the ground plane anyway.

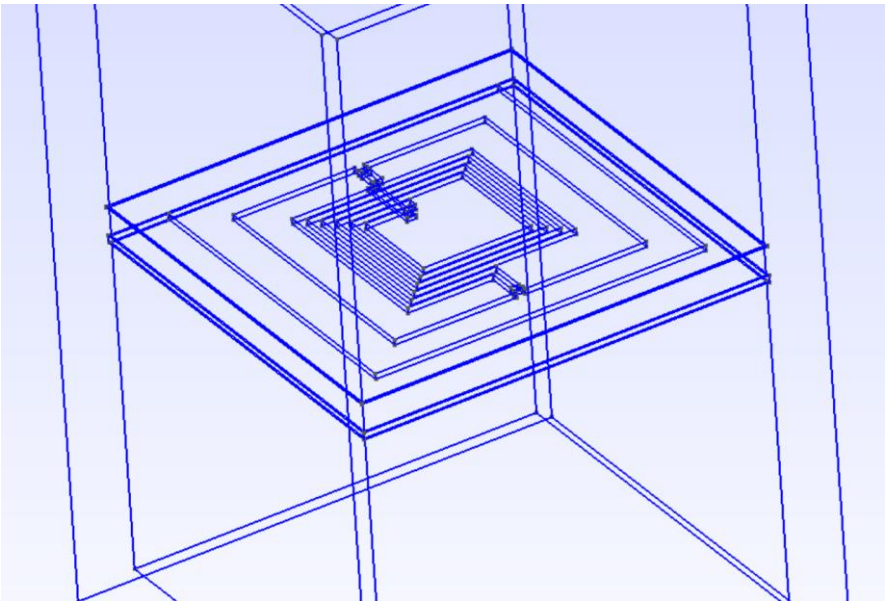


palace_line_noGDS.py

Similar to palace_line_viaport.py but geometry is created by code instead of reading it from GDSII.

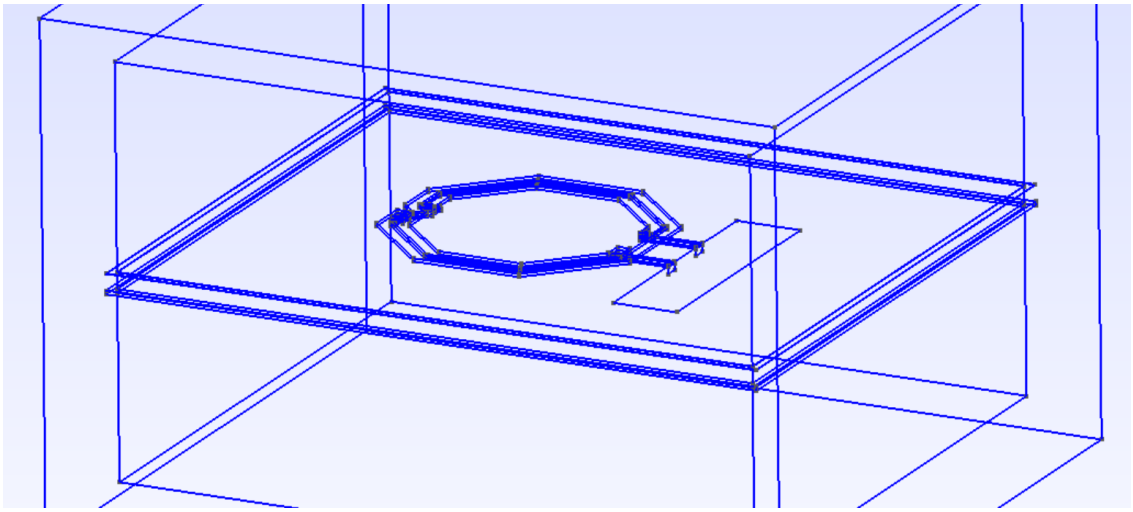
palace_ind_frame.py

This is a 2-port inductor embedded into a metal frame. Geometry from GDSII.

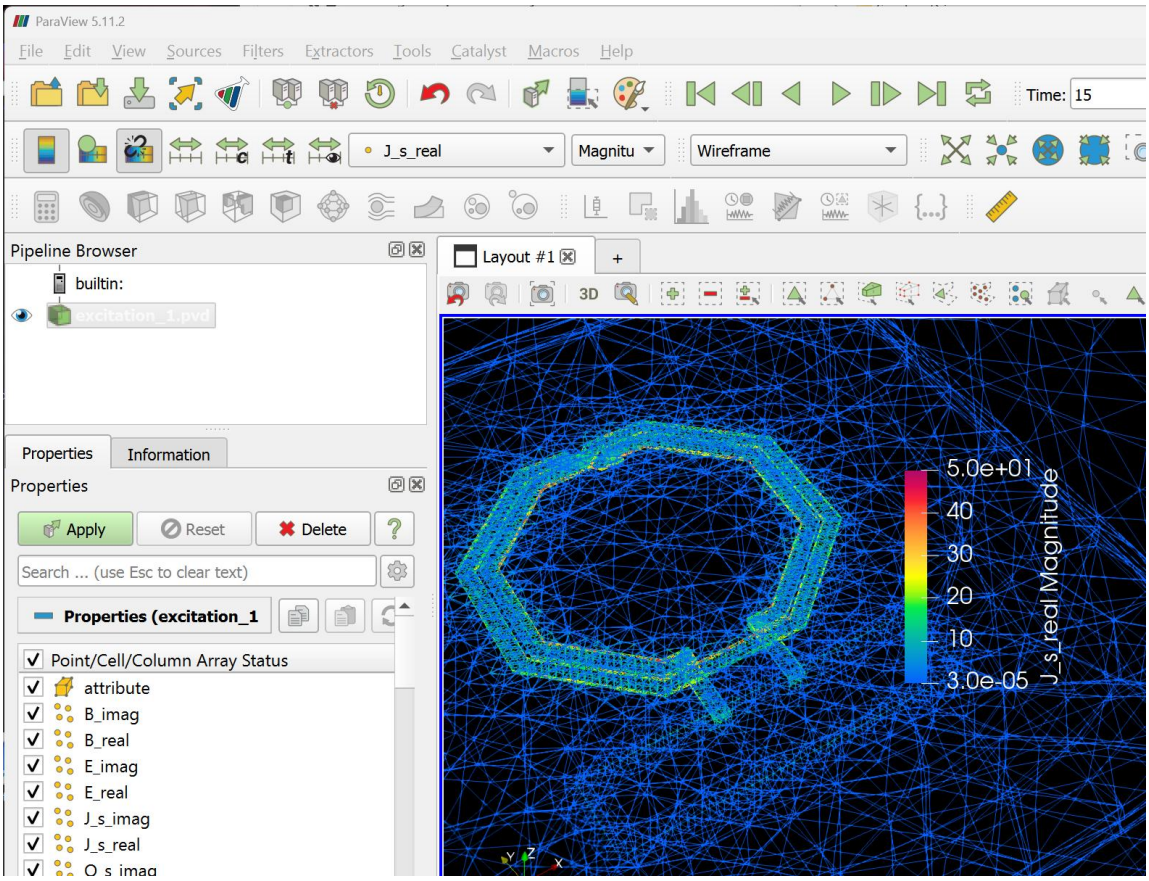


palace_L2n0.py

2-port octagon inductor with via ports down to an artificial metal plane that sits on the surface of bulk silicon.

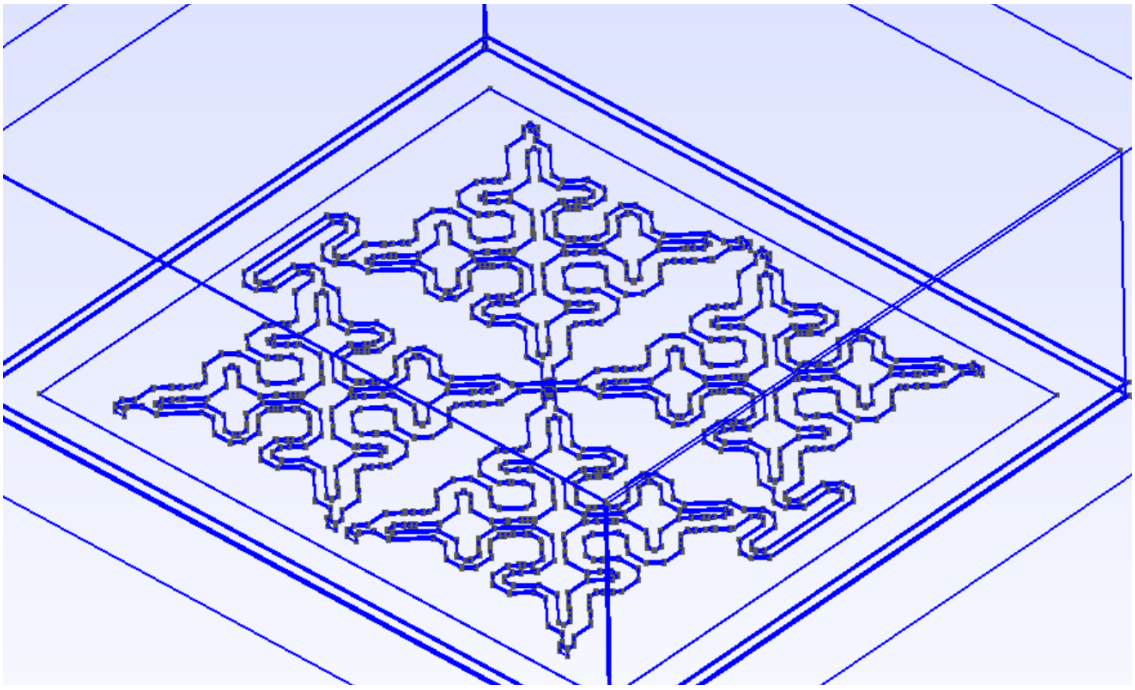


Field dump at a single frequency is enabled in this model.
The Paraview screenshot below shows current density using this file::palace_L2n0_data\output\palace_L2n0\paraview\driven_boundary\excitation_1\excitation_1.pvd



palace_butlermatrix.py

This is quite a big model, the Butler Matrix by Ardavan Rahimian in IHP Open PDK Tapeout July 2025, available at https://github.com/IHP-GmbH/TO_July2025/tree/main/W_Band_Butler_Matrix_IC



Geometry is from GDSII, the model uses a total of 8 via ports between Metal3 and TopMetal2. However, to speed up simulation, only one port excitation is active in this model, and we only get S11,S21,S31,S41,S51,S61,S71 and S81 from this simulation run.

```
simulation_ports = simulation_setup.all_simulation_ports()
# instead of in-plane port specified with target_layername, we here use via port specified with from_
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1, voltage=1, port_Z0=50, sourc
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=2, voltage=0, port_Z0=50, sourc
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=3, voltage=0, port_Z0=50, sourc
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=4, voltage=0, port_Z0=50, sourc
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=5, voltage=0, port_Z0=50, sourc
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=6, voltage=0, port_Z0=50, sourc
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=7, voltage=0, port_Z0=50, sourc
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=8, voltage=0, port_Z0=50, sourc
```

All other port excitations are set to voltage=0 in the model. To get full 8-port S-parameters, we would need to change all port voltages to 1 and re-run the model, so that all excitations are simulated, one after another.

palace_butlermatrix_dump93.py

Same as before, but a field dump setting is added in this model at 93 GHz.

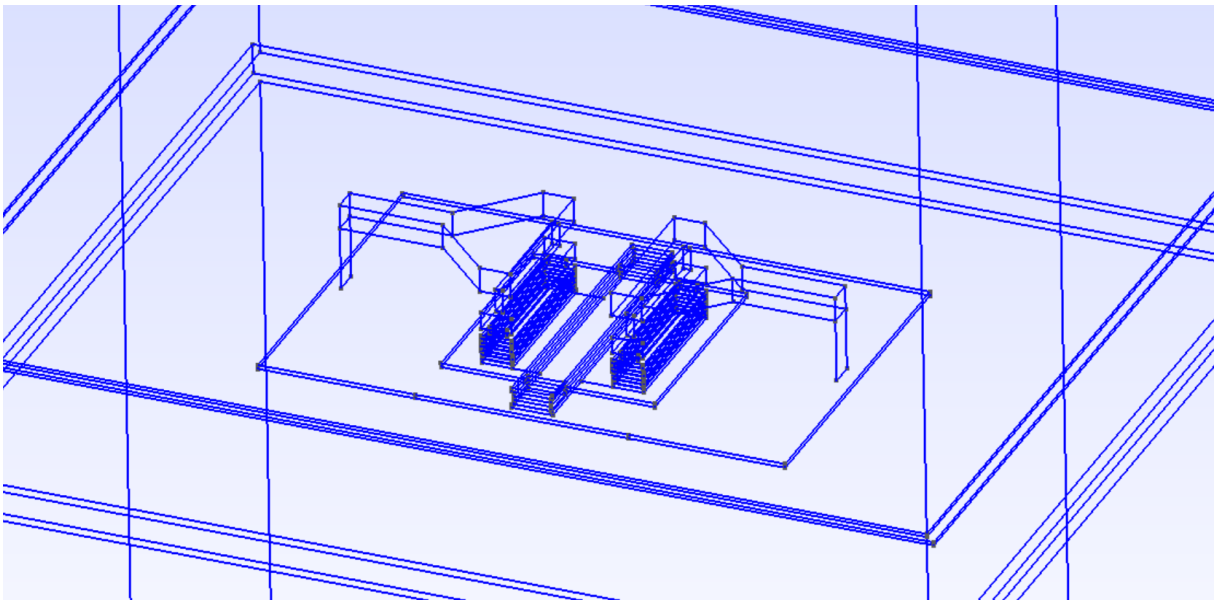
```
settings['fstart']  = 90e9
settings['fstop']   = 95e9
settings['fstep']   = 0.1e9

# settings["fpoint"] = [93e9]  # list of discrete frequencies to be simulated

settings["fdump"] = [93e9]  # save field dump at these frequency points
```

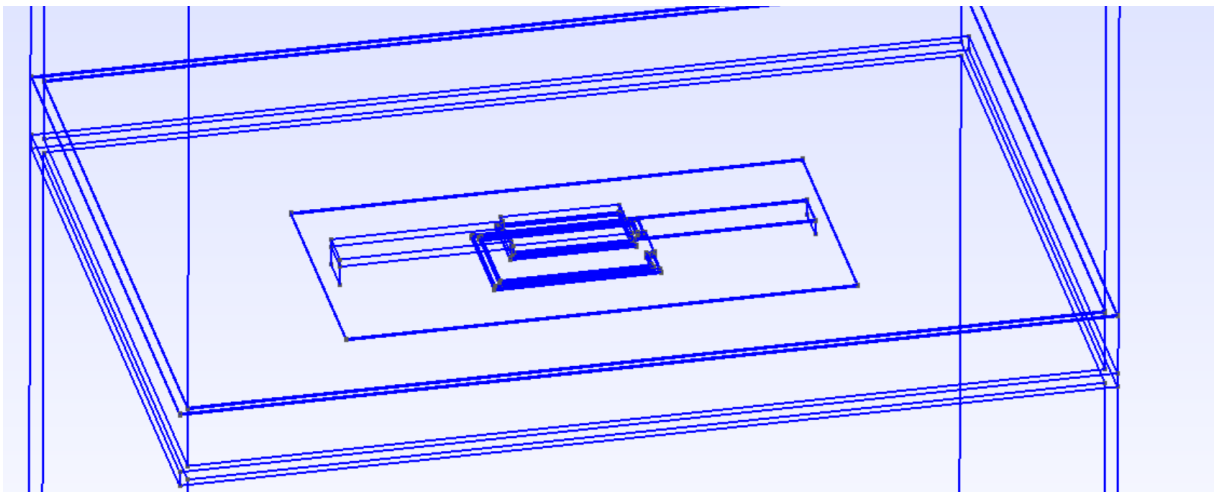
palace_core.py

This is the 4-port “core” model of the 60 GHz medium power amplifier from IHP Analog Academy online course. There are two via ports at the input and output, and two in-plane ports on Metal2 between the common emitter polygon and base/collector coming down from the via stacks.



palace_rfcmim.py

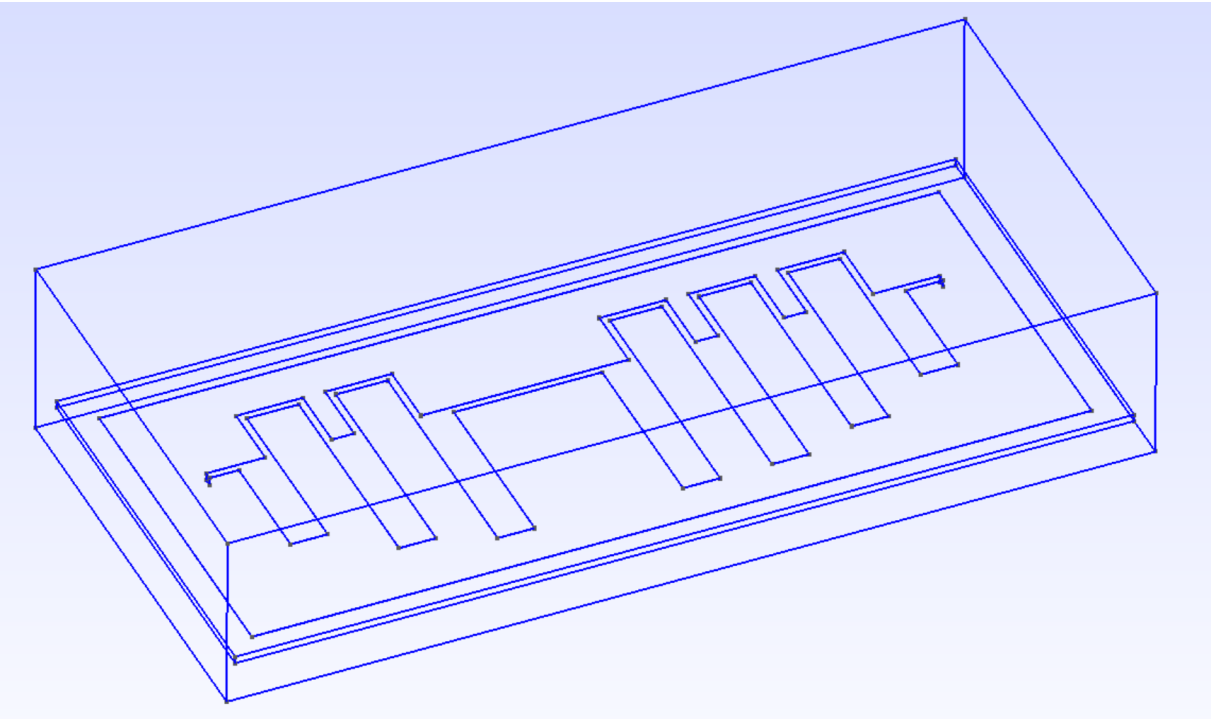
This is 2-port model of an rf_cmim component with some feedline length. Via ports are used at the input and output down to the Metal1 ground plane.



Via arrays are merged by this setting: merge_polygon_size = 2

palace_pcb_lowpass.py

This shows a use case of gds2palace outside the RFIC domain: the layout of a PCB lowpass on RO4003 substrate was imported to klayout and saved in GDSII format.



An XML stackup file was created for this substrate, with layout layers matching the layer numbers used in klayout.

```
▼<Stackup schemaVersion="2.0">
  ▼<Materials>
     <Material Name="Copper" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="3.3e7" Color="00ff00"/>
     <Material Name="RO4003" Type="Dielectric" Permittivity="3.38" DielectricLossTangent="0.0022" Conductivity="0" Color="01e0ff"/>
     <Material Name="AIR" Type="Dielectric" Permittivity="1.0" DielectricLossTangent="0.0" Conductivity="0" Color="d0d0d0"/>
   </Materials>
  ▼<ELayers LengthUnit="um">
    ▼<Dielectrics>
       <Dielectric Name="RO4003" Material="RO4003" Thickness="510"/>
     </Dielectrics>
    ▼<Layers>
       <Substrate Offset="0"/>
       <Layer Name="Bottom" Type="Conductor" Zmin="-17" Zmax="0" Material="Copper" Layer="1"/>
       <Layer Name="Top" Type="Conductor" Zmin="510" Zmax="527" Material="Copper" Layer="10"/>
     </Layers>
   </ELayers>
 </Stackup>
```

Here, setting “margins” only controls the (small) oversize of the dielectrics from the drawing layers. The optional “air_around” setting was used in the simulation model to define the air margins independently, without adding these air layers in the XML stackup.

```
settings['unit']   = 1e-6  # geometry is in MILLIMETER for PCB example
settings['margin'] = 2000    # distance in microns from GDSII geometry boundary to simulation boundary
settings['air_around'] = [1000,1000,1000,1000,2000, 10000] # airbox size to simulation boundary
```