Using OpenEMS with IHP SG13 (Python Interface)

Author: Volker Mühlhaus
Document version: 1.1 of 16 June 2023

Contents

Overview

This document gives an introduction to using OpenEMS with the IHP SG13G2 technology, using the Python scripting interface.

OpenEMS is an Open Source EM simulation tool based on FDTD method. It can be used to analyze layout structures, e.g. inductors, and calculate the corresponding S-parameters.

The vertical stackup of the simulation model is independent of the user layout, it depends on the SG13G2 layer stackup and comes as a predefined technology template. The actual user layout from polygons and vias is then inserted into that template, referencing the SG13G2 material definitions and stackup positions.

Layout import

To simplify building that layout part of the model, a Python script is provided that reads a GDSII file and creates the corresponding layout object in OpenEMS syntax. This layout file can then be combined with the (fixed) technology template.

Layout import using the script can also do via array merging, which combines arrays of closely spaced vias into one large via covering the bounding box.

Meshing

One important aspect of EM simulation is meshing: how to divide the analysis volume into small 3D boxes for the field solver. Reality is continuous, but in simulation we need to reduce complexity and use only a limited number of 3D boxes for solving the actual EM field equations. The finer the mesh, the closer we get to reality, but the price to pay is simulation time and memory requirement.

In this initial version, meshing in the layout region is homogeneous with a fixed mesh size defined by the user. This mesh size must be small enough to resolve relevant details such as line width and gap size, and it must also be small enough to capture skin effect inside conductors.

In many cases, the simulation model includes a margin (empty area) between layout and simulation model boundaries. The mesh in that empty region is created automatically, with less mesh density than the active layout region.

Ports

Similar to other EM solvers used in RFIC EM simulation, ports must be defined where the layout connects to external circuitry.

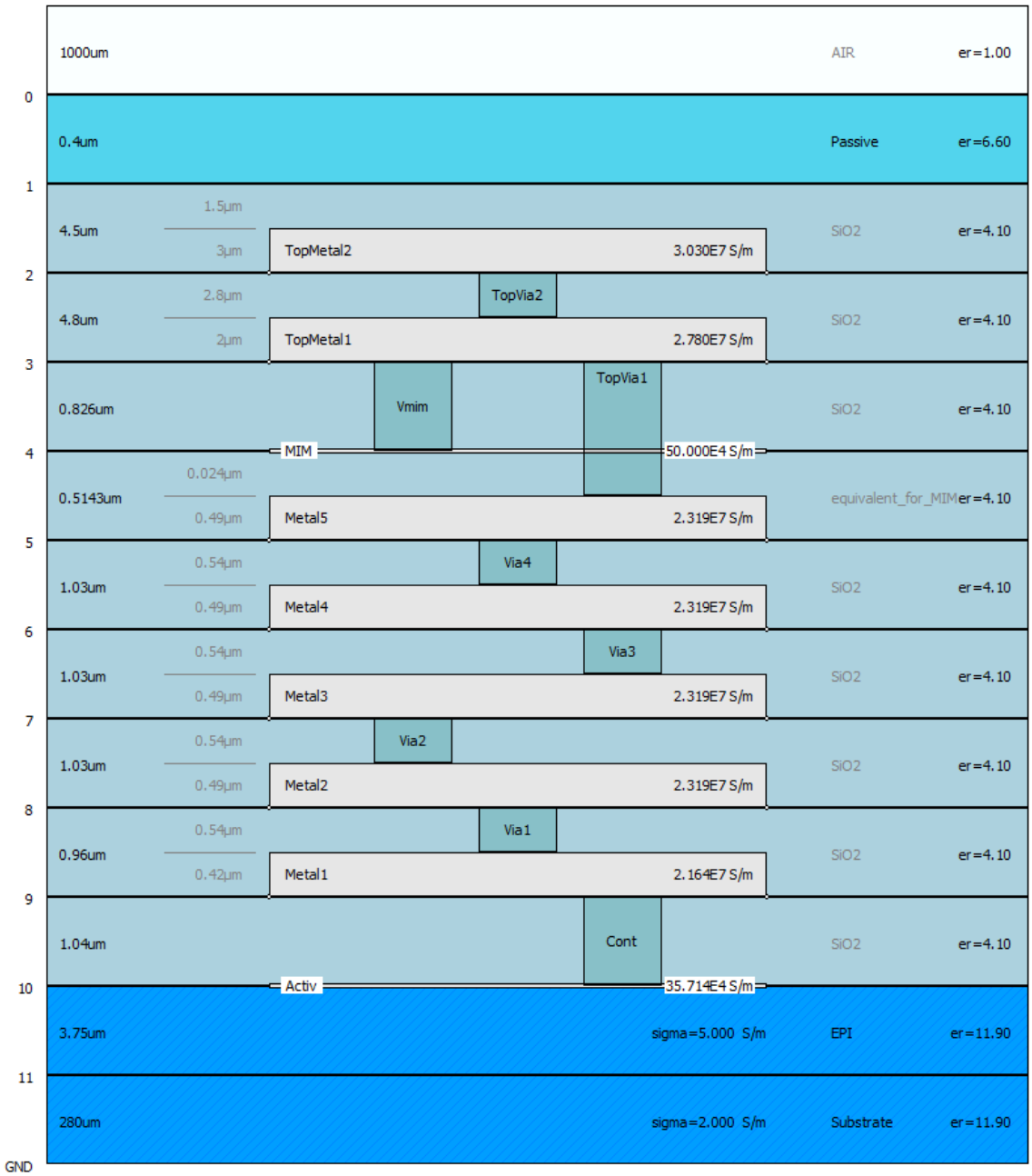
In openEMS, we use lumped ports that can be located in-plane (between polygons on the same layer) or vertical (from one layer up or down). Ports should be electrically small compared to the layout itself, so that we don’t introduce parasitic inductance due to port dimensions.

Ports must be added manually into the model by the user, there is no automation to create ports.

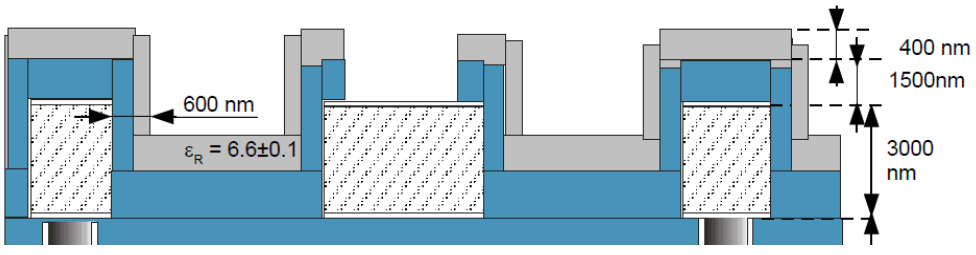
For the inductor example discussed here, a single port is used. This gives frequency-dependent series impedance (effective L and R). One easy way of using that data is to simply model R and L at the frequency of interest in circuit simulation.

SG13G2 stackup cross section

The cross section below shows the EM stackup used for simulation at 300µm total chip thickness.



Passivation and SiO2 above TopMetal2 are not planarized in reality, the planar representation is an approximation for EM modelling only.



Stackup cross section for openEMS model

In openEMS, the stackup is represented by script lines as shown below (not complete):

```
# silicon substrate
Sub = CSX.AddMaterial('Sub', epsilon=11.9, kappa=2)
Sub_thick = 280
Sub_zmin = 0
Sub_zmax = Sub_zmin + Sub_thick

# EPI
EPI = CSX.AddMaterial('EPI', epsilon=11.9, kappa=5)
EPI_thick = 3.75
EPI_zmin = Sub_zmax
EPI_zmax = EPI_zmin + EPI_thick

# SiO2
SiO2 = CSX.AddMaterial('SiO2', epsilon=4.1)
SiO2_thick = 17.73
SiO2_zmin = EPI_zmax
SiO2_zmax = SiO2_zmin + SiO2_thick

(…)

# TopMetal2
TopMetal2_sigma = 3.0300E7
TopMetal2_thick = 3
TopMetal2_zmin  = SiO2_zmin + 11.23
TopMetal2_zmax  = TopMetal2_zmin + TopMetal2_thick
TopMetal2 = CSX.AddMaterial('TopMetal2', kappa=TopMetal2_sigma)

# TopMetal1
TopMetal1_sigma = 2.7800E7
TopMetal1_thick = 2
TopMetal1_zmin  = SiO2_zmin + 6.43
TopMetal1_zmax  = TopMetal1_zmin + TopMetal1_thick
TopMetal1 = CSX.AddMaterial('TopMetal1', kappa=TopMetal1_sigma)

(…)
```

These material definitions are identical for all models using SG13G2 technology, independent of the user drawn layout.

For metal and via layers, the user drawn layout will reference these definitions. For dielectric and semiconductor layers, a box made from these materials will be added in the model code, with dimensions calculated from the bounding box of user drawn layout.

Simulation frequency, meshing, boundaries

The simulation **frequency range** must be entered into the simulation model as needed.

In addition, the simulation **boundaries** around the analysis volume must be defined. For non-radiating structures we can use PEC boundaries, that are perfect electric conductor walls. To avoid an interaction between metal side walls and the device under test, we add some spacing at the sides and above. For inductor simulations, that spacing on the sides is +/- one inductor diameter, so that the total box size in xy-plane is 3 inductor diameters. The bottom of the substrate is usually placed on a metal boundary (PEC) with no extra spacing.

```
############ simulation settings ############

unit = 1e-6 # specify everything in um
refined_cellsize = 1  # mesh resolution in area with polygons from GDSII

fstart = 0
fstop  = 30e9
numfreq = 401  # number of frequency points (has no effect on simulation time!)

energy_limit = -50    # end criteria for residual energy
Boundaries   = ['PEC', 'PEC', 'PEC', 'PEC', 'PEC', 'PEC']  # xmin xmax ymin ymax zmin zmax

eps_max = 11.9 # maximum permittivity in model, used for calculating max cellsize

wavelength_air = (3e8/unit)/fstop
max_cellsize = wavelength_air/(sqrt(eps_max)*20) # max cellsize is lambda/20 in medium
```

Variable **refined_cellsize** defines the mesh size in xy-plane for user-drawn layout parts, value in microns. This must be small enough to resolve relevant details such as line width and gap size, and it must also be small enough to capture skin effect inside conductors. Simulation of thick metal conductors in openEMS uses a general material definition that has no built-in skin effect model, so the meshing must be small enough that the field solver can “see” the effect of field decay inside conductors.

Variable **max_cellsize** is the maximum mesh size to be used in xyz-direction, it is automatically calculated from frequency range assuming a maximum mesh size of 1/20 wavelength in Silicon.

Variable **energy_limit** is the limit when FDTD simulation is finished because results are considered accurate enough. The default of -50dB means that residual energy inside the analysis volume is down by 50dB from the initial port input signal, because signals have either been dissipated by lossy materials or left the analysis volume at the ports (transmitted or reflected).

Geometries from GDSII

Geometries can be converted from GDSII format to openEMS syntax using script *gds2pythonpoly.py*

The script is run from the command line, the only parameter is the GDSII filename. For example,

```
python3 pythonpoly.py L2n0.gds
```

will create an output file *L2n0_polygons.py* which can then be inserted into the model template. The layer table inside the script defines what GDSII layer numbers and purposes are evaluated, and map the corresponding material names for the output file.

Below is an excerpt from *L2n0_polygons.py* to show what the geometries in openEMS Python syntax look like:

```
# Cell ('L_2n0_simplify", 10 polygons, 0 paths, 2 labels, 0 references)

pts_x = np.array([])
pts_y = np.array([])
pts_x = r_[pts_x, 22.200]
pts_y = r_[pts_y, 0.000]
pts_x = r_[pts_x, 34.200]
pts_y = r_[pts_y, 0.000]
pts_x = r_[pts_x, 34.200]
pts_y = r_[pts_y, 57.000]
pts_x = r_[pts_x, 22.200]
pts_y = r_[pts_y, 57.000]
pts = np.array([pts_x, pts_y])
TopMetal1.AddLinPoly(priority=200, points=pts, norm_dir ='z', elevation=TopMetal1_zmin,
length=TopMetal1_thick)

pts_x = np.array([])
pts_y = np.array([])
pts_x = r_[pts_x, -23.230]
pts_y = r_[pts_y, 272.000]
pts_x = r_[pts_x, -9.985]
pts_y = r_[pts_y, 272.000]
pts_x = r_[pts_x, 5.015]
pts_y = r_[pts_y, 257.000]
pts_x = r_[pts_x, 23.230]
pts_y = r_[pts_y, 257.000]
pts_x = r_[pts_x, 23.230]
pts_y = r_[pts_y, 269.000]
pts_x = r_[pts_x, 9.985]
pts_y = r_[pts_y, 269.000]
pts_x = r_[pts_x, -5.015]
pts_y = r_[pts_y, 284.000]
pts_x = r_[pts_x, -23.230]
pts_y = r_[pts_y, 284.000]
pts = np.array([pts_x, pts_y])
TopMetal1.AddLinPoly(priority=200, points=pts, norm_dir ='z', elevation=TopMetal1_zmin,
length=TopMetal1_thick)

(...)

pts_x = np.array([])
pts_y = np.array([])
pts_x = r_[pts_x, 11.850]
pts_y = r_[pts_y, 257.620]
pts_x = r_[pts_x, 22.600]
pts_y = r_[pts_y, 257.620]
pts_x = r_[pts_x, 22.600]
pts_y = r_[pts_y, 268.370]
pts_x = r_[pts_x, 11.850]
pts_y = r_[pts_y, 268.370]
pts = np.array([pts_x, pts_y])
TopVia2.AddLinPoly(priority=200, points=pts, norm_dir ='z', elevation=TopVia2_zmin,
length=TopVia2_thick)
```

These polygon calls reference layer/material names and vertical stackup positions defined earlier in the stackup section.

Geometry bounding box information is found at the end of the file, this data is later used to create a uniform fine mesh in the active layout part.

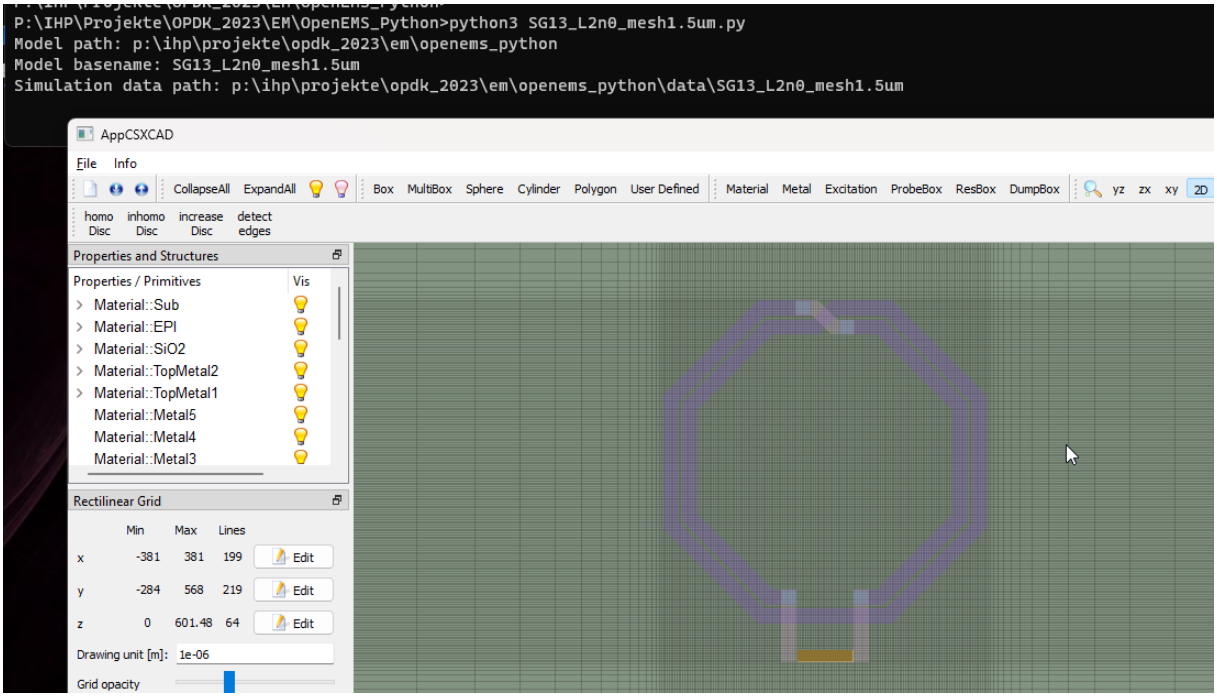
Ports

There is no port information included in the GDSII file, all ports must be added manually in the openEMS model code. For the inductor example shown later in this document, a single port is defined between the two terminals, resulting in 1-port data (*.s1p).

Impedance measured into the port can be converted to equivalent series resistance and series inductance. The example Python script prints out that data at one user defined frequency.

Running the model file in Python

For the octagon inductor example discussed in this document, run python3 <modelname. py> from the command line. This will first start the model viewer so that you can check the model and mesh. When the model viewer is closed, simulation starts.




```
P:\IHP\Projekte\OPDK_2023\EM\OpenEMS_Python>python3 SG13_L2n0_mesh1.5um.py
Model path: p:\ihp\projekte\opdk_2023\em\openems_python
Model basename: SG13_L2n0_mesh1.5um
Simulation data path: p:\ihp\projekte\opdk_2023\em\openems_python\data\SG13_L2n0_mesh1.5um

Running simulation ...
 ----------------------------------------------------------------------
 | openEMS 64bit -- version v0.0.35-108-gc651cce
 | (C) 2010-2023 Thorsten Liebig <thorsten.liebig@gmx.de>  GPL license
 ----------------------------------------------------------------------
        Used external libraries:
                CSXCAD -- Version: v0.6.2-123-gc29742b
                hdf5   -- Version: 1.12.0
                          compiled against: HDF5 library version: 1.12.0
                tinyxml -- compiled against: 2.6.2
                fparser
                boost  -- compiled against: 1_72
                vtk -- Version: 8.2.0
                       compiled against: 8.2.0

Create FDTD operator (compressed SSE + multi-threading)
Multithreaded operator using 16 threads.
```

The actual simulation model for openEMS is an XML file that is created by the Python code after all material and geometries and simulation settings are defined.

The code snippet shown below shows that part of the *.py model file:

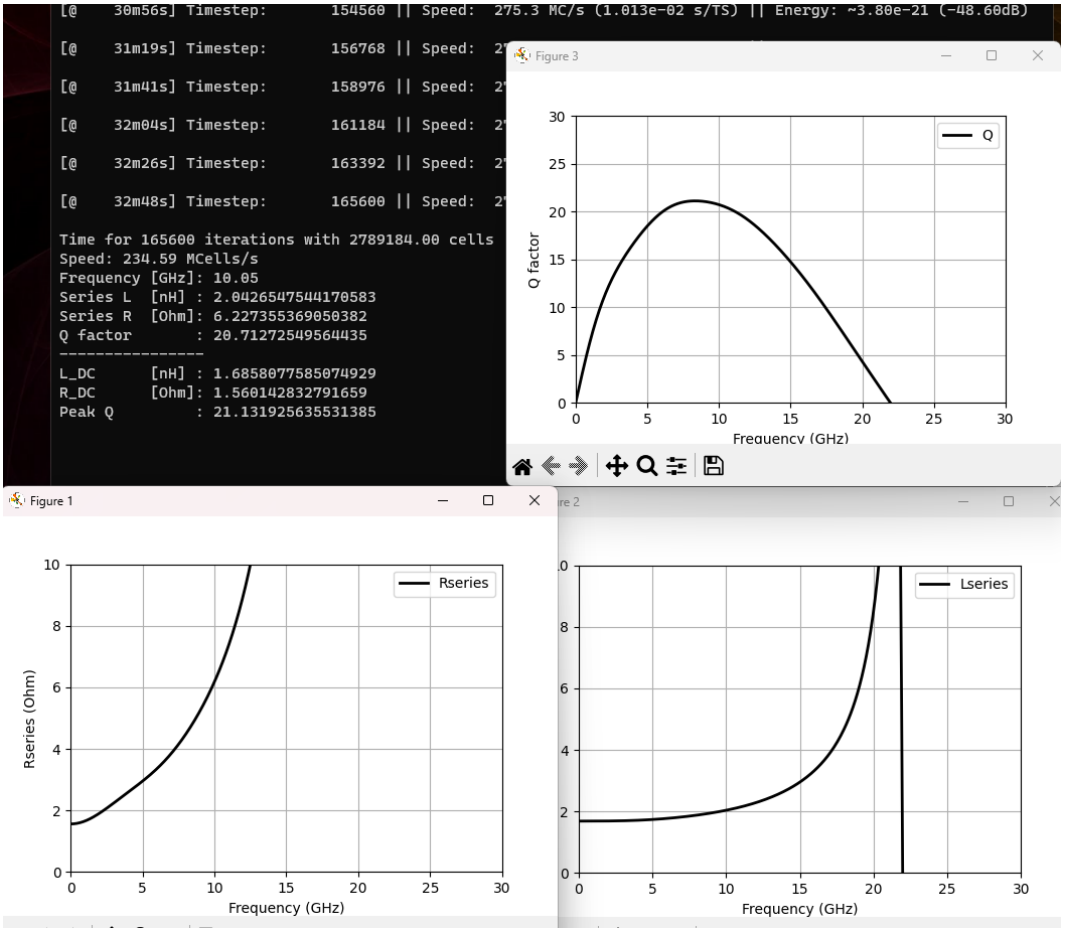
```
if not postprocess_only: # skip model preview and simulation
  from CSXCAD import AppCSXCAD_BIN
  os.system(AppCSXCAD_BIN + ' "{}"'.format(CSX_file))

if not preview_only:  # start simulation
    if not postprocess_only:
        print("Running simulation ...")
        FDTD.Run(sim_path, verbose=1)

    ### Post-processing and plotting ###
    f = np.linspace(fstart,fstop,numfreq)
    port.CalcPort(sim_path, f)
```

Variables preview_only and postprocess_only are defined at the beginning of the script. By setting one of these values to True, you can preview the model and mesh withoutout starting the simulator, or you can postprocess existing simulation results (for plotting) without re-running the full EMsimulation.

```
# preview model/mesh only?
# postprocess existing data without re-running simulation?
preview_only = False
postprocess_only = False
```

Plotting results and creating S-parameter output

When simulation is finished (which can take 60 minutes for the exampel shown here, depending on mesh size and computer speed), postprocessing will read results and the plots defined in the script are created. Also, the script calculates some inductor parameters for information.



At the end of the script, S11 results are saved in Touchstone *.s1p format.

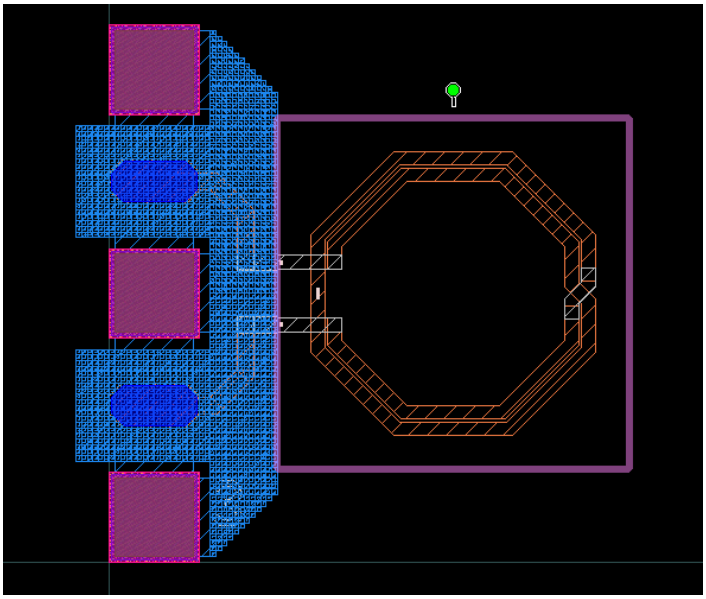
Simulation model and resulting data, including S-parameters (here: *.s1p) are created in a directory “data” below the model file. If simulation data already exists, you can created additional plots by changing the script and re-running with variable “postprocessing_only = True”

Testcase: Octagon Inductor L3_2n0

To verify the simulation flow and accuracy of results, a 2nH octagon inductor in SG13G2 technology was modelled for which reliable measurements are available.

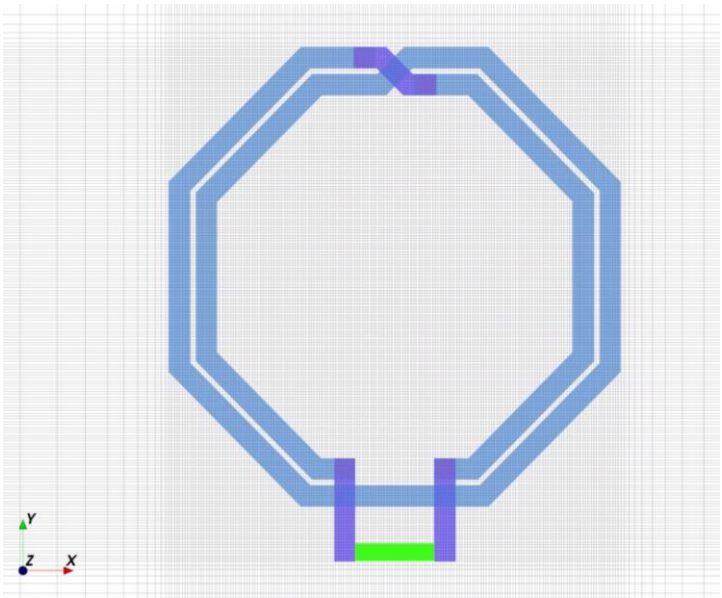
Geometry parameters: N=2, w=12µm, s=3µm, Di=200µm, feedline spacing (center to center): 56.4µm

At only 3µm spacing between the turns, capacitance between the thick conductor metals will have an impact on self resonance frequency.



Measurements are de-embedded using GSGSG calibration standards to remove the feed structure, so that the inductor model contains only the inductor itself (pink boundary in screenshot above).

For simulation, an in-plane port is added between the feed lines on TopMetal1. In the AppCSXCAD preview of the simulation model below, that port is shown in green.



From this 1-port simulation we get the impedance (series L, R and Q factor) for differential operation. To compare that with measurements, 2-port measurement results are also converted to differential 1-port data between the inductor terminals.

Via arrays –effect of via array merging

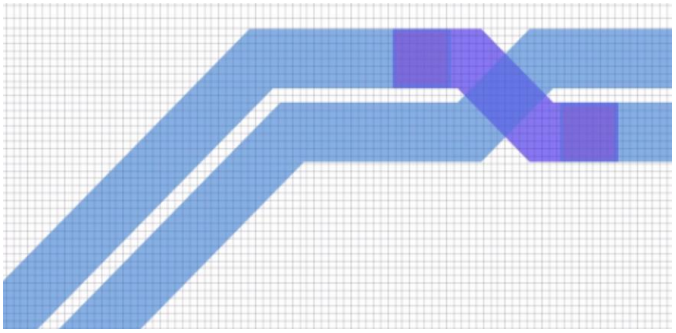
Vias in the GDSII file had already been merged prior to GDSII export, so that we simulate the bounding box of via arrays instead of all individual vias. Note that merging vias is usually not required for simulation time in FDTD method, if the mesh size is on the order of via size anyway.

For accuracy, via array merging is not critical here, because each of the via arrays has 36 vias of 1.1 Ohm each = 0.03 Ohm resistance per via array. In the inductor path we have 4 via arrays in series, for a total of 0.12 Ohm from all via arrays. Compared to the total inductor series resistance of a few Ohms, we will not notice the small error from via array merging when comparing results.

Mesh size

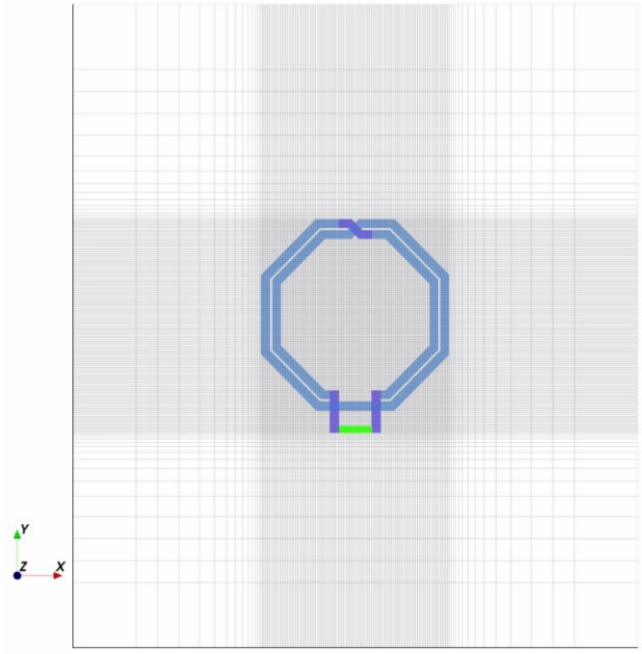
For mesh size, we have to consider two aspects: accurate geometry sampling and skin effect.

At 3µm gap width and 12µm line width, a mesh size of 1.5µm will capture the geometry without creating incorrect gap width along the xy-axis. However for the diagonal lines, we will have some staircasing effects that can be reduced by using a smaller mesh size.



The second aspect is skin depth: for Aluminium, skin depth is approximately 0.5µm at 25 GHz and 0.25µm at 100 GHz. To accurately capture the skin effect where fields are pushed to the outer edges of conductor cross section, we need sufficiently fine mesh density.

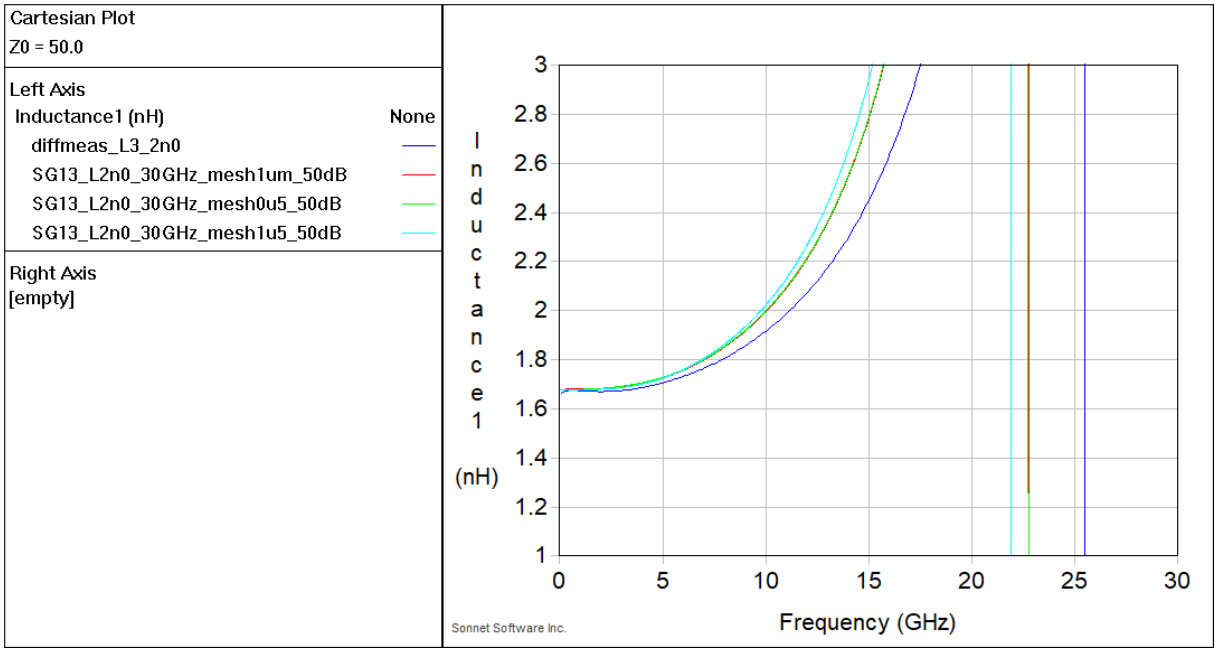
Different mesh sizes in the xy-plane were used for simulation, and compared to measurements. Mesh in z-direction was not changed here, although mesh refine for skin effect would need to cover all directions.



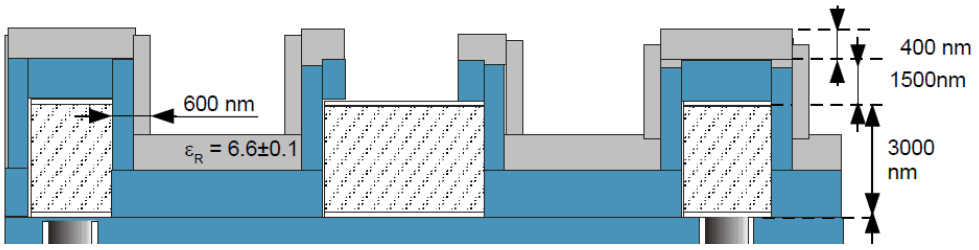
Inductance simulation vs. measured

For inductance, simulation results are almost identical between 1.0µm and 0.5µm mesh size. Only at 1.5µm we see a small change in SRF towards lower frequencies, which might be explained by staircasing effects in the diagonal lines that change capacitance between the turns.

Inductance agrees well to measurements, except for a difference in SRF:
simulated SRF at 0.5µm mesh is 22.7 GHz, compared to 25.5 GHz in measurement.



This might be explained by the planar passivation used in simulation (all gap between turns completely filled with SiO2), whereas the true passivation in hardware is conformal, with some air between the turns. This would explain a higher SRF in measurement, due to lower capacitance between the turns.

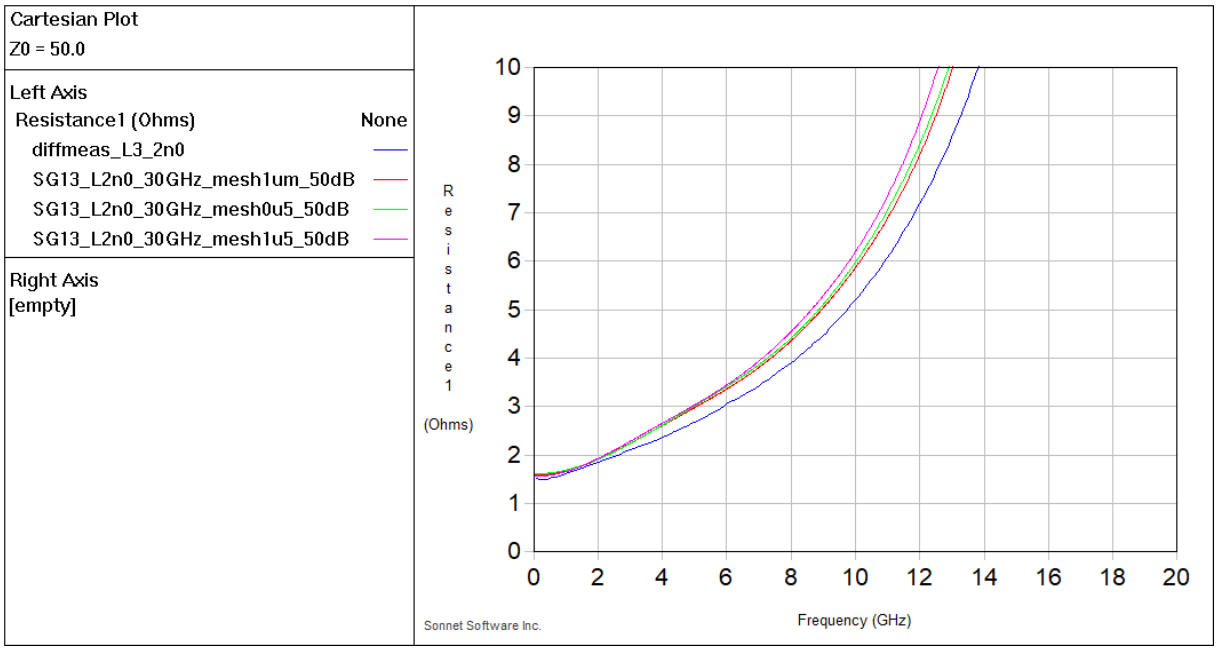


At only 3µm gap between the turns, this difference in inductor sidewall capacitance is visible in this inductor testcase.

Resistance simulation vs. measured

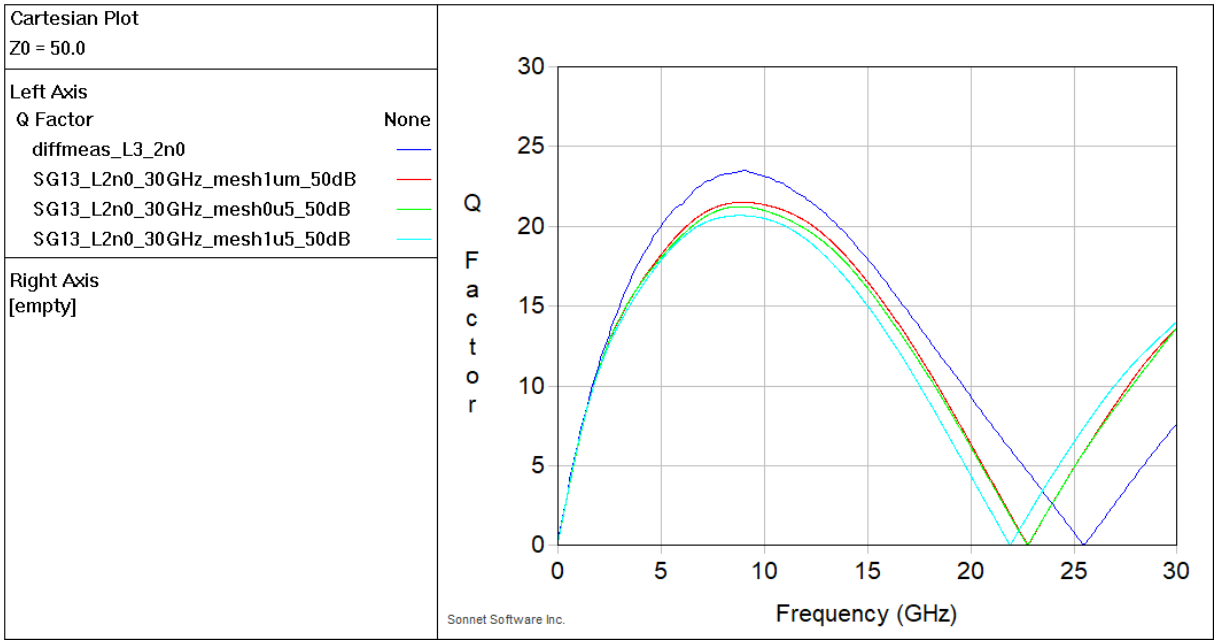
For effective series resistance, simulation results are very close between 1.0µm and 0.5µm mesh size, and 1.5µm is not much different.

Measured series resistance agrees very well at low frequency, and then shows slightly less increase towards high frequencies. Much of this can be explained by the difference in SRF discussed earlier.



Q factor simulation vs. measured

Comparing the Q factor, we see good agreement between measured and simulated data, except for the difference in SRF discussed earlier.



The frequency of peak Q agrees very well: 8.9 GHz simulated va. 9 .0 GHz measured. Peak Q of 23.5 measured vs. 21.2 simulated @ 0.5µm mesh size is also good agreement, if we consider that very small changes in S-parameters means significant changes in Q factor at these large Q values.

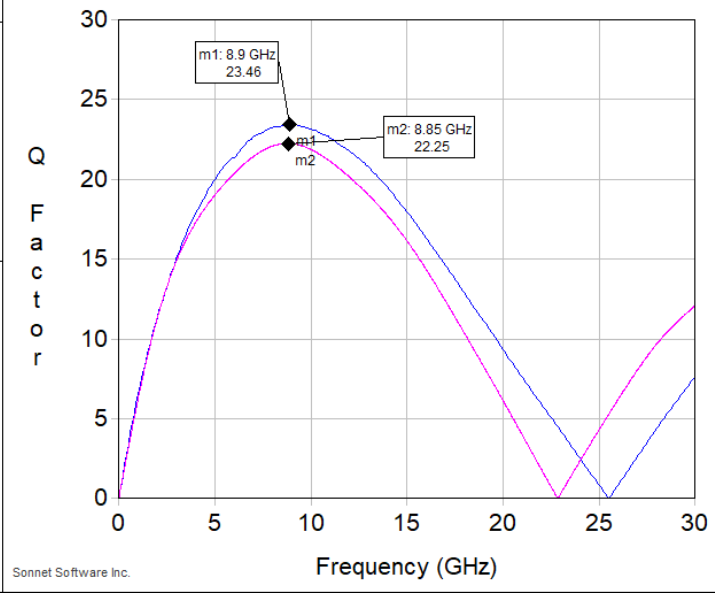
When the vertical mesh (z-direction) is also refined, at 0.5µm mesh size in xy-direction, the peak Q value increases from 21.2 to 22.3 at 8.9 GHz. That is really close to the measured peak Q of 23.5.

SRF does not change with this refined vertical mesh, as expected.

Cartesian Plot
Z0 = 50.0

Left Axis
Q Factor None
diffmeas_L3_2n0
m1: 8.9 GHz
23.46
SG13_L2n0_30GHz_mesh0u5_50dB_finerZ2
m2: 8.85 GHz
22.25

Right Axis
[empty]



Simulation time

Using a fine mesh for simulation is more accurate, but requires more simulation time.

One reason is the larger **total number of mesh cells**, but there is one other influence in FDTD: this method calculates the propagation of signals through the model in time domain, and the **timestep** used for calculation must be small enough to capture propagation in the smallest mesh cell. **If one mesh cell in the model is really small, that determines the time step used for the entire calculation of all cells.**

If the mesh is very dense in one place (even if that is only one small mesh cell) that determines the time step for the entire model, regardless of mesh density anywhere else in the model.

Memory required for simulation

Memory size is usually not a problem, because FDTD method is very memory efficient and required RAM scales linear with total number of mesh cells.

Number of simulation frequencies

The number of simulation frequencies does **not** have an effect on simulation time or memory. This is because the EM simulation in time domain (FDTD) is the same, and only FFT in postprocessing changes.

Number of ports

For FDTD, we usually excite only one port at a time, so we need multiple simulation runs to get the **full** S-matrix of a multi-port device. In this case, multiple XML files must be created by the script, each with a different excited port and all other ports not excited.

Note that we do not always need to excite all ports: if we are interested in S11 and S21 only, it is enough to excite only port 1, and leave the other ports as “receiving” ports only.

The next chapter provides some multi port excitation examples where multiple ports are excited, to obtain the full [S] matrix for S-parameter export.

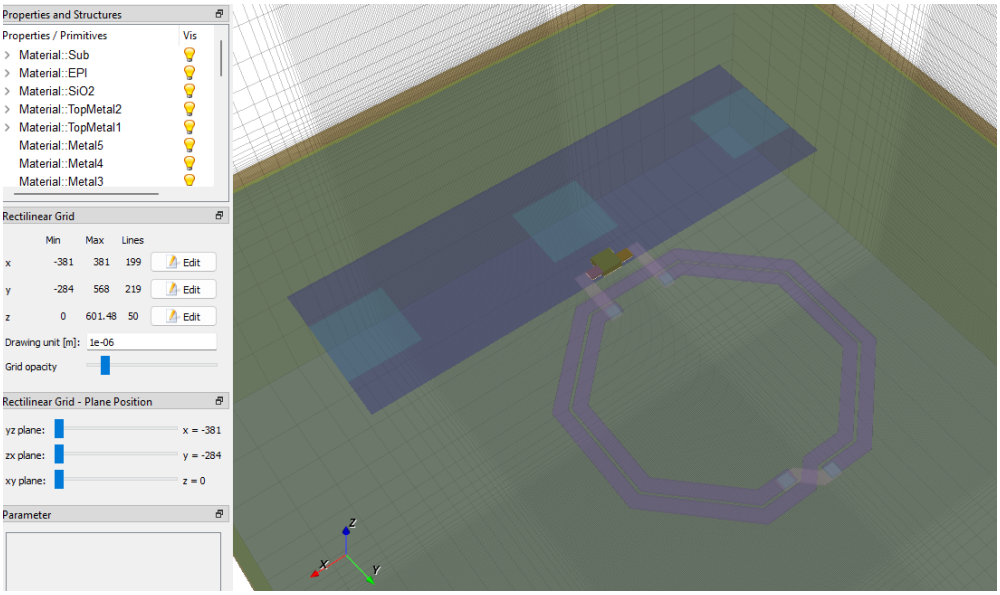
Multi-port simulation examples

2-port inductor simulation with reference node at substrate

Inductor models discussed earlier in this document use a single port between the inductor terminals, which represents differential mode parameters. However, there is no information on substrate shunt effect in those simulation results.

To capture both shunt and series path, a two port model can be build that has one port on each inductor terminal, with common ground node connected to the substrate.

The 2-port model **SG13_L2n0_2port_groundpolygon_mesh1.5um.py** is shown below:



In openEMS (FDTD) that 2-port model requires excitation at both ports, one after another. In the code, function *createSimulation()* is called twice, once for each port, to run these two excitations. Only with both results we get the full 2x2 element [S] matrix.

For comparison to 1-port results, the code below also extracts series L and series R by converting the 2-port data into differential 1-port data. That calculated 1-port data is the same as directly simulating the 1-port model in openEMS (which is also available as an example).

Microstrip line

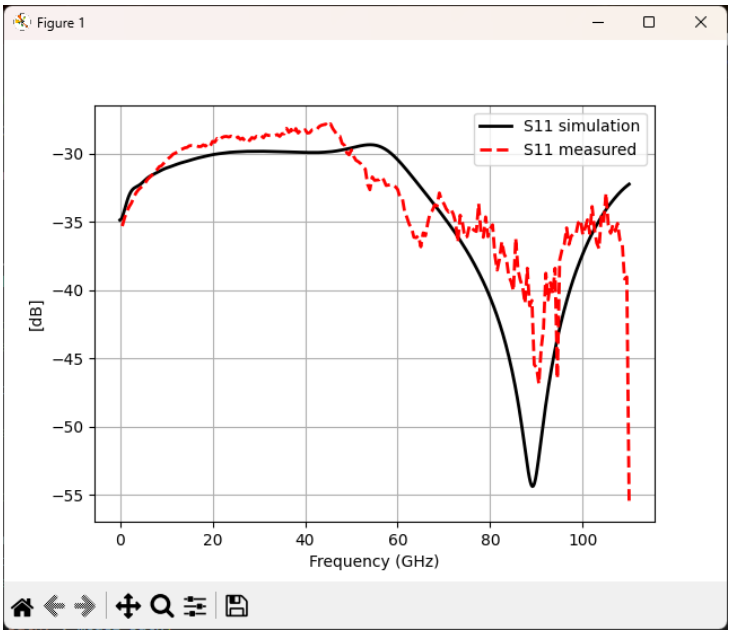
Model **SG13_line_2port_TM2overM1_880um_PML.py** represents an RFIC microstrip line in SG13G2 technology, simulated using two ports. The line is on TopMetal2 over Metal1 ground, line width w=15um for 50 Ohm. Dimensions are set by parameters in the geometry section code.

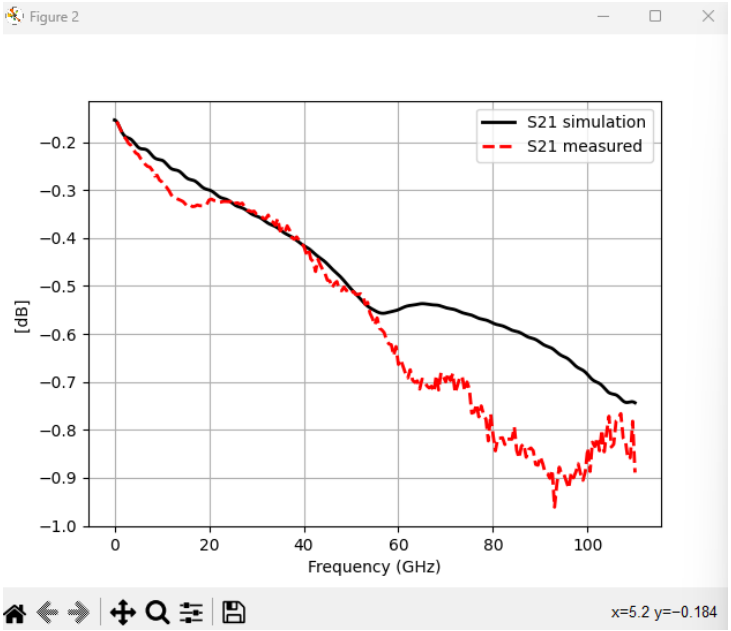
FDTD method excites one port at a time: excite port 1 to get S11 and S21, excite port 2 to get S22 and S12.

In this model, function createSimulation() is called twice, once for each port, to run these two excitations. Only with both results we get the full 2x2 element [S] matrix so that we can create an S2P output file.

If you want to save time, and don't need the reverse direction S-params, you can set variable full_2port = False which only calculates port 1 excitation.This gives S11 and S21 only, and the S2P file will not be created.

For this transmission line, simulation results are compared to measurements. Measurement data for the line is obtained by de-embedding the pad sections from raw measurement.





For these comparisons, some utility functions are defined in **s2p_utils.py**