# Using OpenEMS with IHP SG13: Python Workflow Version 2

Author: Volker Mühlhaus
Document version: 1.2 of 10 February 2025

## Contents

## What is new in this version?

This document gives an introduction to OpenEMS with the IHP SG13G2 technology, using a completely redesigned new Python workflow. This modular workflow is much easier to use than the initial version, it can read GDSII input files with no need for prior conversion and provides some automatic meshing capabilities. Ports can now be created from GDSII polygons on special layers, instead of coding their position in the Python code. The EM stackup is defined in an XML file, which can be created from ADS Momentum substrate files using a Python script. The entire model code is much cleaner and easier to navigate.

## System Requirements

This workflow is based on the Python workflow for OpenEMS, please refer to
https://www.openems.de/
and https://docs.openems.de/python/install.html#python-linux-install

If you have trouble to build OpenEMS for Linux, please check out the OpenEMS forum
https://github.com/thliebig/openEMS-Project/discussions
Running the Windows version is easier because pre-built binaries are available.

In addition to OpenEMS, the Python module gdspy must be installed.

## What is openEMS and FDTD?

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.

Some basic properties of the FDTD method:

- The entire simulation model **volume** is meshed into many small cubes. This process is also called meshing.
- External **boundaries** around the simulation box are metal or absorbing or symmetry plane.
- The model is excited at **one** port with a wideband gaussian pulse, running a **time domain** simulation of the field propagation through model until energy in model is below a user defined threshold. The port voltages and currents over time are stored to disk.
- Using an **FFT**, the time domain voltages are processed to get the frequency domain response. This gives **wideband S-parameters**, but only **partial for that one signal path**, e.g. S11, S21, S31, … for excitation at port 1.
- If necessary, this can be repeated, exciting one port after another, to get the **full** S-matrix

As an example, we simulate a 4 port coupler with excitation at port 1. Simulation gives us the resulting voltages at all ports. From this we can calculate incident + reflected voltage at port 1 and transmitted voltage at ports 2,3,4.

Using FFT we can then calculate S-Params S11, S21, S31, S41

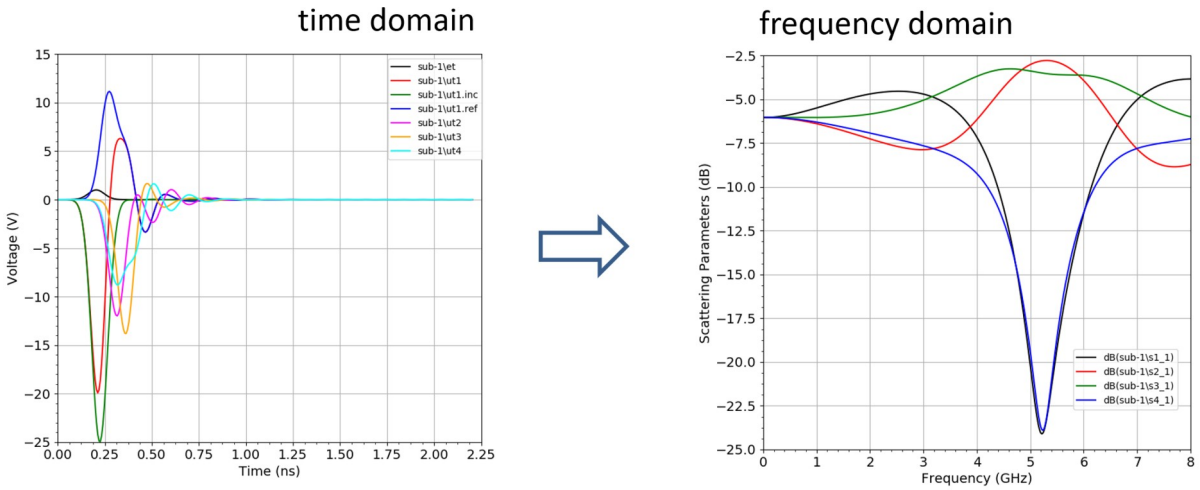



*Figure 1: S-parameter from FDTD time domain EM*

It is important to understand the difference between Method of Moments (used in Momentum or Sonnet) and the FTDT method (Finite Difference in Time Domain):

In MoM, the solver simulates on frequency after another, and we get all S-parameters for that one frequency. To cover a frequency range, we need to repeat simulation at more frequencies.

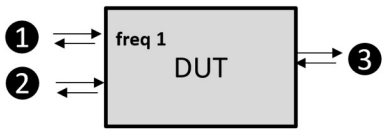
In FDTD, the solver simulates one port excitation with a wideband gaussian pulse, and by using FFT we get the wideband S-parameters for the signal path from this excitation port to all other ports. To get the full [S] matrix with all possible signals paths, we need to repeat simulation with excitation at all other ports, one after another.

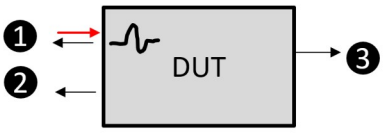
**Method of Moments, FEM**

- Frequency domain solver
- Full S-Matrix for one frequency

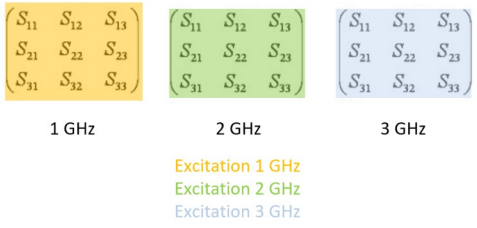
**FDTD**

- Time domain solver, pulse excitation
- One column of S-matrix for complete frequency range




FFT time domain -> frequency domain



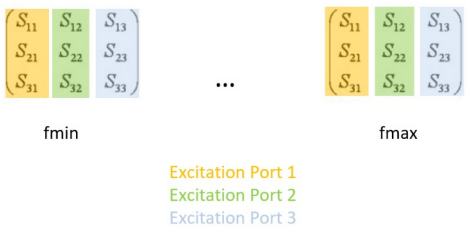




*Figure 2: Comparison between both EM simulation methods*

## Time steps

With FDTD being a **time domain** method, the number of frequency points that we get from FFT has no effect on simulation time, because that is a postprocessing step only. Instead, the **time step** used in simulation determines total simulation time.
This time step depends – among other factors – on the smallest mesh cell used in discretizing the model: Every timestep must capture the field propagation at every mesh cell. Timesteps must be small enough to resolve the wave propagation even at the smallest mesh cell.
This means that very fine mesh cells should only be used for a good reason, to resolve detail that make a difference for results, like small geometry details or thin dielectric layers, or to resolve skin effect in conductors.

The end of the simulation period is reached if the energy in the simulation model drops below a certain convergence limit (by reflection, transmission or dissipation) or when a maximum number of time steps is reached (finish without reaching convergence)

If the simulation terminates before the energy limit is reached, results might show some ripple in the data and/or glitches in the data at low frequency. This might happen for high Q resonances, for example, which have a very slow energy decay.

## Simulation volume and boundaries

The FDTD simulation volume is always a cube of finite size, with defined boundary conditions on all six sides. To simulate an open environment for antennas, absorbing boundaries are used. For other models, perfect conductors (PEC) on the side walls are often used.

In the simulation model, the choice of boundary condition is done separately for all six sides, so that we can also simulate an antenna that has a PEC bottom side.

Mesh lines run across the entire simulation volume, with no local mesh refinement.

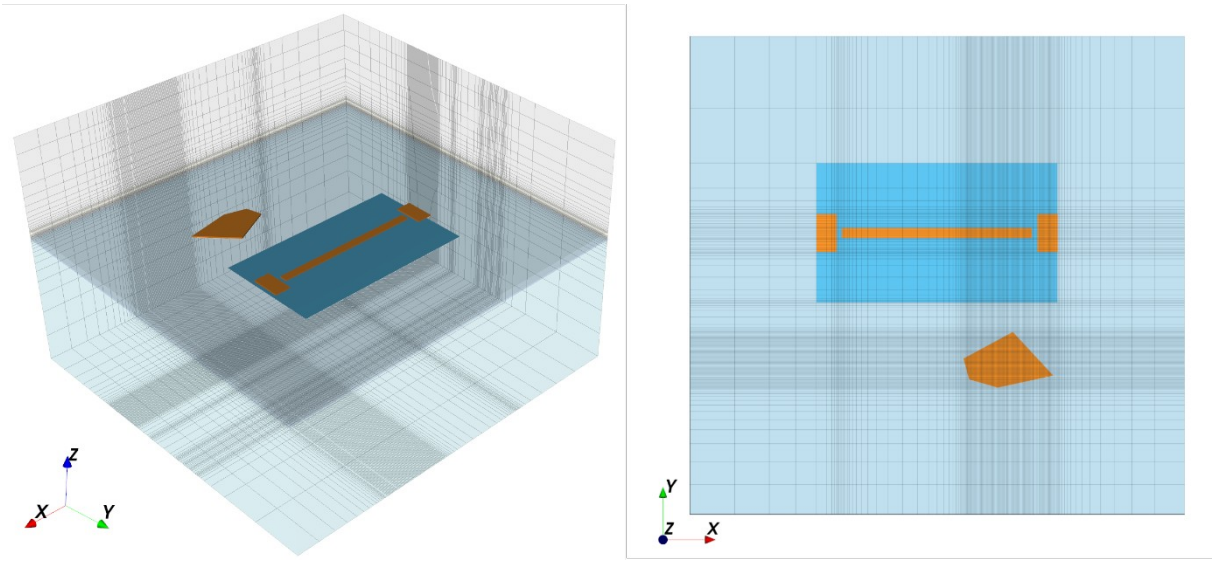



*Figure 3: 3D view and top view of a simulation model with mesh lines*

Meshing is discussed later in this document in more detail. The general rule is: create a mesh that is fine enough to capture all **relevant** field detail, but don't mesh too fine because finer mesh means more cells and slower simulation time. After running a few models and looking at results, you will have an idea how to determines a proper mesh size.

## Overlapping materials

Using OpenEMS, metals can be inserted into a dielectric block with no need for boolean operations. Instead, each material has a **priority** that defines which material "wins" in case of overlapping materials. In the Python workflow described here, metals are automatically set to highest priority, then vias, then dielectrics.

# The OpenEMS Python workflow for IHP OPDK

The workflow presented in this document is based on the Python API for openEMS.

Compared to native OpenEMS Python model, this workflow tries to encapsulate the “routine” stuff that repeats in every model into utility modules, giving some level of abstraction for the model code that the user works with. Also, technology stackup and GDSII layout can be imported with very few lines of code. In addition, two predefined meshing options are available..

The goal is to create a clean model code that is easier to create and understand than regular OpenEMS Python models. However, in those places where models are unique and the user needs flexibility to code his own special settings, this is still possible.

## Technology

Dielectrics and semiconductors in the simulation model are independent of the drawn layout, and come from a technology template. This template also includes other technology information like layer definitions for metal layers and via layers, and their position in the stackup.

All this data comes as a predefined technology template in XML format. Note that substrate thickness is defined in the technology template, so different templates are required for different substrate thickness.

```
<Stackup schemaVersion="2.0">
  <Materials>
    <Material Name="Activ" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="357141.0" Color="00ff00"/>
    <Material Name="Metal1" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="21640000.0" Color="39bfff"/>
    <Material Name="Metal2" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="23190000.0" Color="ccccd9"/>
    <Material Name="Metal3" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="23190000.0" Color="d80000"/>
    <Material Name="Metal4" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="23190000.0" Color="93e837"/>
    <Material Name="Metal5" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="23190000.0" Color="dcd146"/>
    <Material Name="TopMetal1" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="27800000.0" Color="ffe6bf"/>
    <Material Name="TopMetal2" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="30300000.0" Color="ff8000"/>
    <Material Name="TopVia2" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="3143000.0" Color="ff8000"/>
    <Material Name="TopVia1" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="2191000.0" Color="ffe6bf"/>
    <Material Name="Via4" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="1660000.0" Color="deac5e"/>
    <Material Name="Via3" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="1660000.0" Color="9ba940"/>
    <Material Name="Via2" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="1660000.0" Color="ff3736"/>
    <Material Name="Via1" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="1660000.0" Color="ccccff"/>
    <Material Name="Cont" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="2390000.0" Color="00ffff"/>
    <Material Name="Passive" Type="Dielectric" Permittivity="6.6" DielectricLossTangent="0.0" Conductivity="0" Color="a0a0f0"/>
    <Material Name="SiO2" Type="Dielectric" Permittivity="4.1" DielectricLossTangent="0.0" Conductivity="0" Color="fffcad"/>
    <Material Name="Substrate" Type="Semiconductor" Permittivity="11.9" DielectricLossTangent="0" Conductivity="2.0" Color="01e0ff"/>
    <Material Name="EPI" Type="Semiconductor" Permittivity="11.9" DielectricLossTangent="0" Conductivity="5.0" Color="294fff"/>
    <Material Name="AIR" Type="Dielectric" Permittivity="1.0" DielectricLossTangent="0.0" Conductivity="0" Color="d0d0d0"/>
    <Material Name="LOWLOSS" Type="Conductor" Permittivity="1" DielectricLossTangent="0" Conductivity="1E10" Color="ff0000"/>
    <Material Name="MIM_equiv" Type="Dielectric" Permittivity="16.87" DielectricLossTangent="0.0" Conductivity="0" Color="ff0000"/>
  </Materials>
  <ELayers LengthUnit="um">
    <Dielectrics>
      <Dielectric Name="AIR" Material="AIR" Thickness="300.0000"/>
      <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="280.0000"/>
    </Dielectrics>
    <Layers>
      <Substrate Offset="283.75"/>
      <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="LBE" Type="via" Zmin="-283.7500" Zmax="0.0000" Material="AIR" Layer="157"/>
      <Layer Name="SUBGND" Type="via" Zmin="-3.75" Zmax="0" Material="LOWLOSS" Layer="210"/>
      <Layer Name="MIM_DK" Type="conductor" Zmin="5.5800" Zmax="5.6800" Material="MIM_equiv" Layer="36"/>
      <Layer Name="MIM" Type="conductor" Zmin="5.6800" Zmax="6.4303" Material="TopVia1" Layer="36"/>
    </Layers>
  </ELayers>
</Stackup>
```

*Figure 4: IHP stackup SG13.XML used for OpenEMS workflow*

The file format for this technology template was developed for this workflow, it is not a standard file format used by other EM tools.

## Layout and ports

The layout is read directly from a GDSII file. This file needs some special treatment before using it: **the best way to create ports for this openEMS OPDK workflow is to add them to the GDSII file** on special layers (recommended: layers 201 and above).

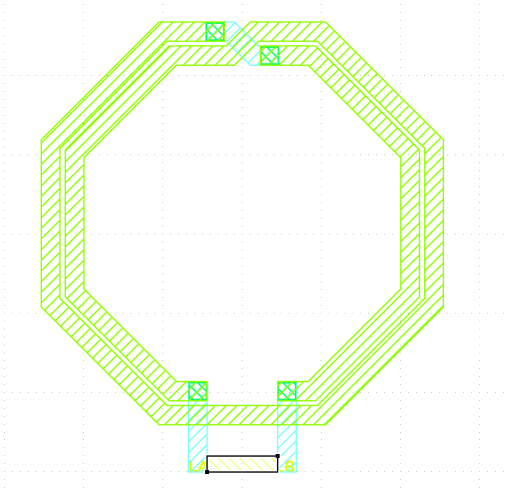


*Figure 5: GDSII layout with port shape drawn on layer 201*

The reason for drawing port polygons on special layers, instead of using the pin purpose, is that PDK pins often have the wrong shape for using them as OpenEMS ports. We need to create lumped ports that are drawn **between** polygons, to close the conductor loop, and this is fundamentally different from the use of PDK pins.

Layout import from GDSII can do optional via array merging, which combines arrays of closely spaced vias into one large via covering the bounding box. In addition, GDSII import can pre-process the layout to handle complex shapes with holes.

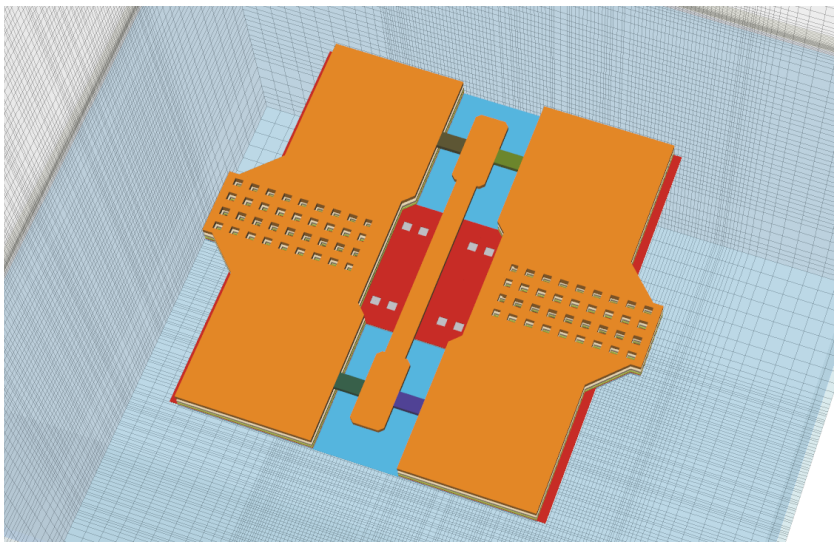


*Figure 6: Complex layout with many holes imported with optional GDSII pre-processing, GSG ports created from 2 polygons per port on layers 201-204*

## 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 version of the workflow, two meshing algorithms are available:

- constant mesh size within the GDSII drawing area, at a spacing defined by the user
- automatic meshing based on detecting polygon edges, with some automation to handle diagonal lines and  mesh lines that are too close.

In any case, the user defines target 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.

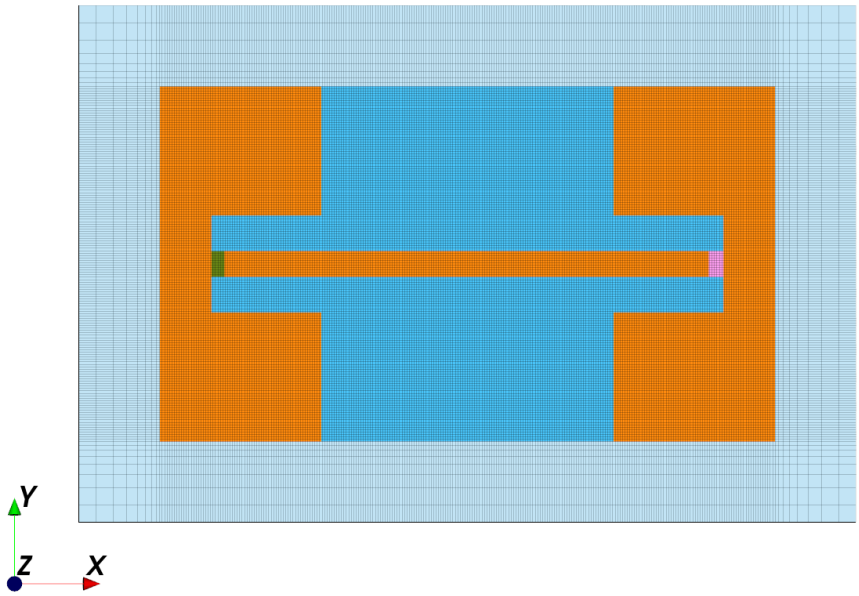



*Figure 7: Fixed meshing inside bounding box of GDSII polygons, graded mesh outside*

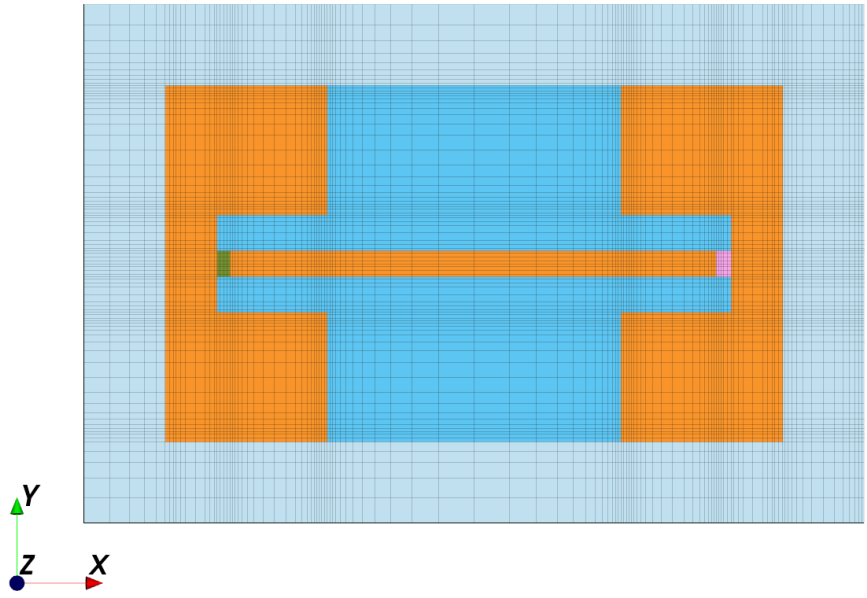



*Figure 8: Automatic meshing*

## Minimum configuration

The screenshot below shows a minimum configuration, which consists of the XML technology stackup, the GDSII layout, one simulation model file (here named run_inductor_diffport.py) and the utility modules with all the "behind the scenes" code that you don't need to modify.

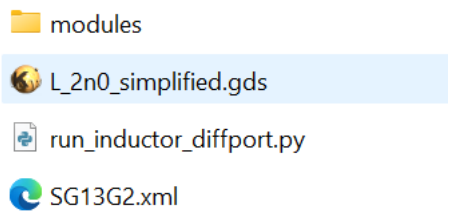



*Figure 9: Minimum configuration*

Compared to plain openEMS Python code, there is a lot of abstraction in the model file for this workflow, to keep the model clean and easy to navigate. All the code that users don't need to touch is moved to utility modules in the modules folder.

To showcase the use for many different scenarios, like complex port configurations or antenna simulation, additional examples are available which are not shown in the minimum file set above. All these example models are named run_...

To run the inductor model, type "python3 run_inductor_diffport.py" at the command line. The actual action then depends on the values of *preview_only* and *postprocess_only* in your model code, as explained in the next chapter.

In the beginning, you should run with *preview_only=True,* until you are sure that the model and mesh is what you want to simulate. This will bring up the 3D model preview, but not start the actual EM simulation yet.

## Missing modules

The workflow code uses some Python modules like the gdspy reader for GDSII. If you get an error message regarding missing modules, like gdspy, you need to install them using pip:

```
pip3 install gdspy
```

## File locations

Please keep all your model files together with the XML stackup and the modules folder in one directory, and make sure that you have write permissions there. Keeping the files together, without mixing files from different workflow releases, will minimize issues when some workflow components are updated in the future.

After running at least one simulation model code, a folder named "output is created, which contains simulation results and the model in XML format for inspection in the AppCSXCAD 3D viewer.

If you do not have write permission in the model folder, the output folder will be created in the temp directory, defined by your TEMP environment variable.

## Simulation model file for inductor example

To create a new model file, it is recommended to start from an existing model file that matches your use case, and then modify the configuration to your needs.

Below we will look at the simulation model for an inductor model with one EM port between the terminals (differential mode, resulting in S1P output).

### Workflow settings

The first lines configure the workflow: When you start working on a model, you want to visually check the model in the AppCSXCAD 3D viewer, look at ports and mesh lines etc. At this stage, you want to set the preview_only switch to True, so that simulation does not start yet.
When you are satisfied with the model preview, run the model script again with preview_only and postprocess_only set to False, so that EM simulation will start running after you close AppCSXCAD.

If postprocess_only is True, the script will skip simulation, so that you can re-run data plots of previously simulated models.

```
# ======================= workflow settings ================================

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

### Input files and path settings

The next section defines the **input files** for GDSII input and EM stackup from XML.

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

gds_filename = "L_2n0_simplified.gds"   # geometries
XML_filename = "SG13G2.xml"               # stackup

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

# 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)
```

It also defines if **GDSII preprocessing** is enabled, which might be necessary for layouts with complex holes and cutouts.

The next lines configure the **model simulation path** (default is in the directory where you also store the simulation script) and the **name of the output directory** (default is the base name of the simulation script).

If you want to change the simulation directory to another place, e.g. some tmp directory, you can do that by changing these code lines.

With these settings, the simulation will create a folder “output/run_inductor_diffport_data” in your directory where the simulation model is stored, and from where you run the simulation.[1]

In that data directory, you will find a subdirectory “sub-n” where n is the number of the port excitation, e.g. port 1 excitation.

After simulation is finished, you will find these files in the “sub-1” directory:


*Figure 10: Simulation output for inductor example with port 1 excitation*

L_2n0_simplified.xml is the simulation model that is sent to the AppCSXCAD 3D viewer just before simulation is started, or if you set “preview_only = True” in your model code.
You can also start AppCSXCAD manually and then load this XML file to visualize the model and related mesh lines. This is useful especially for new users, who are not very familiar with OpenEMS yet: you can look at your own model and also at the examples.

Files port_it_1 and port_ut_1 are time domain voltage and current at port 1 from running the EM simulation. The model code will evaluate these files to get frequency domain S-parameters.

If your model codes includes Touchstone *.s1p file export, this will also be stored here.

## Simulation settings

The next section defines **simulation settings** like frequencies, boundaries, the target mesh size for fixed or automatic meshing and the energy limit when EM simulation is considered finished (converged).

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

unit   = 1e-6   # geometry is in microns
margin = 200    # distance in microns from GDSII geometry boundary to simulation boundary

fstart = 0
fstop  = 30e9
numfreq = 401

refined_cellsize = 1.0  # mesh cell size in conductor region

# choices for boundary:
# 'PEC' : perfect electric conductor (default)
# 'PMC' : perfect magnetic conductor, useful for symmetries
# 'MUR' : simple MUR absorbing boundary conditions
# 'PML_8' : PML absorbing boundary conditions
Boundaries = ['PEC', 'PEC', 'PEC', 'PEC', 'PEC', 'PEC']

cells_per_wavelength = 20   # how many mesh cells per wavelength, must be 10 or more
energy_limit = -50          # end criteria for residual energy (dB)
```

1 If the model directory is not writable, the data folder will be created in the TEMP directory.

Regarding frequencies, note that FDTD method simulates with a wideband pulse, and the frequency domain results are obtained using FFT as a **post-processing** step. The **number of frequency points** specified here has **no** effect on simulation time, it is pure post-processing of the FDTD results in time domain. It is perfectly fine to simulate 401 or more frequency points, there is no speed advantage if you specify less points!

The **boundaries** are listed in this order: xmin, xmax, ymin, ymax, zmin, zmax.

Choices for each boundary are perfect electric conductor (PEC), perfect magnetic conductor (PMC), simple absorbing sheet (MUR) and perfectly matched layer (PML_8).
For fast simulation speed, use PEC if there is no (relevant) radiation from your device under test.
For antennas, use PML_8 for best absorbing properties at the boundary.

The **margin** value in simulation settings defines the distance to the simulation boundary. All stackup materials are extended to the sides by that distance, up to the simulation boundary.

For inductor models, a margin equal to the inductor diameter is considered enough, so that the metal side walls (PEC) surrounding the simulation area don't distort the local inductor fields.

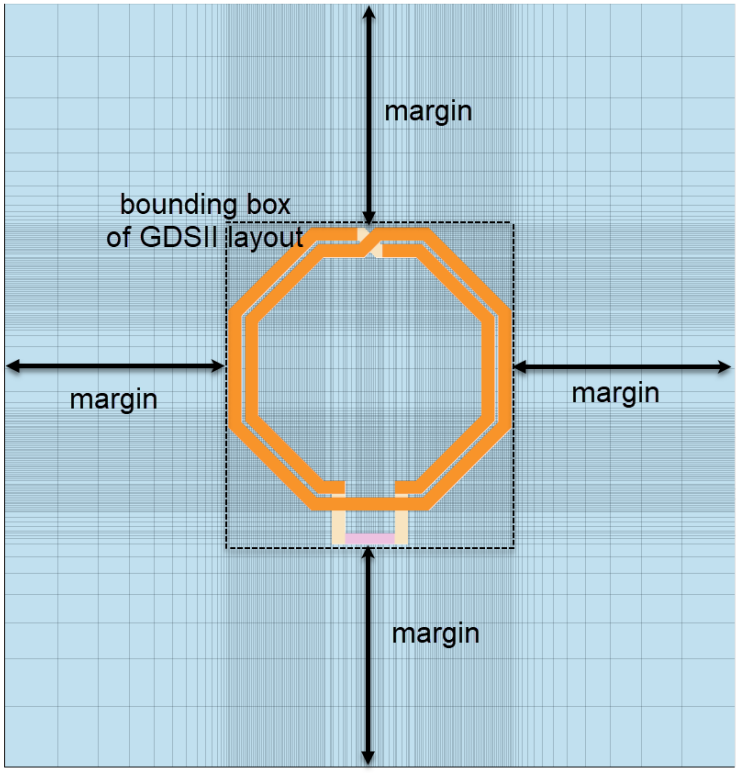



*Figure 11: Margin defined in simulation settings*

For structures like a transmission line over ground, a smaller distance will be ok because there is not much field leakage. To absorb possible weak radiation, a MUR absorbing boundary can be used at the top cover of the simulation box.[2]

The margin value does not apply to the top and bottom boundary (z direction).
The bottom distance is zero by default, the top distance is also zero and only include the air layer as defined in the XML stackup.

For antennas, special settings can be applied, with an extra air layer on all six sides, as explained later in this document.

2 With MUR boundary, simulation did not converge for some RFIC examples. If you notice this, switch to PML_8 absorbing boundaries.

## Port definitions

The next section defines **EM ports**.

```
# ports from GDSII Data, polygon geometry from specified special layer
# note that for multiport simulation, excitations are switched on/off in simulation_setup.createSimulation below
simulation_ports = simulation_setup.all_simulation_ports()
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1, voltage=1, port_Z0=50, source_layernum=201,
                                                           target_layername='TopMetal1', direction='x'))
```

You will notice that this looks different from the original openEMS port definition.

The code defines a simulation_ports object to manage all ports, and then adds port number 1 with amplitude 1 V/m and 50 Ohm port impedance, which is defined on GDSII layer 201 and goes to target layer “TopMetal1” in the EM stackup. The port is oriented in x direction.

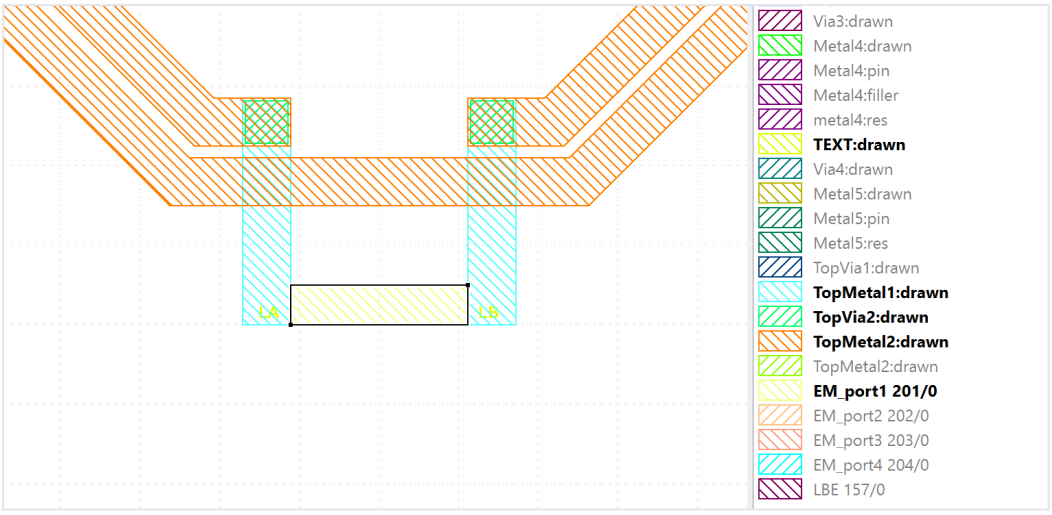



*Figure 12: Port layout defined in GDSII file on layer 201*

The reason for drawing port geometries directly into the GDSII file is that it not only simplifies the model code, it also avoids possible precision errors in defining the port co-ordinates.

The port can be mapped to a metal layer as shown here, with x or y orientation, but it can also be mapped to a via layer and used in z direction, if necessary. This gives maximum flexibility in creating EM ports.

The port direction also supports reverse polarity, by adding a minus sign to the direction. This can be required to place the reference terminal on the correct side, as shown in the example below.

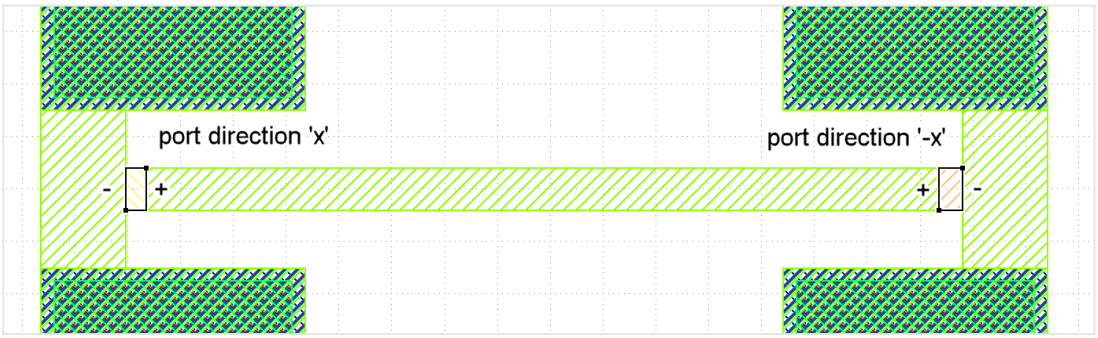



*Figure 13: Example for polarity in port direction*

In some cases, we also need to create “composite” ports, e.g. to model a wafer probe interface with one signal line and two grounds.

For a transmission line with such a GSG port on both ends, the port section would look like this:

```
Z0 = 50
simulation_ports = simulation_setup.all_simulation_ports()
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1, voltage=1, port_Z0=2*Z0, source_layernum=201, \
                                                           target_layername='TopMetal2', direction= 'y'))
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=2, voltage=1, port_Z0=2*Z0, source_layernum=202, \
                                                           target_layername='TopMetal2', direction='-y'))
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=3, voltage=1, port_Z0=2*Z0, source_layernum=203, \
                                                           target_layername='TopMetal2', direction= 'y'))
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=4, voltage=1, port_Z0=2*Z0, source_layernum=204, \
                                                           target_layername='TopMetal2', direction='-y'))
```

The GSG port on each end of the line consists of 2 port definitions each, with one port in y direction and the other port in -y direction, to have the same effective polarity. And because they are effectively in parallel, we specify 2*Z0 as port impedance, to have Z0 in total.

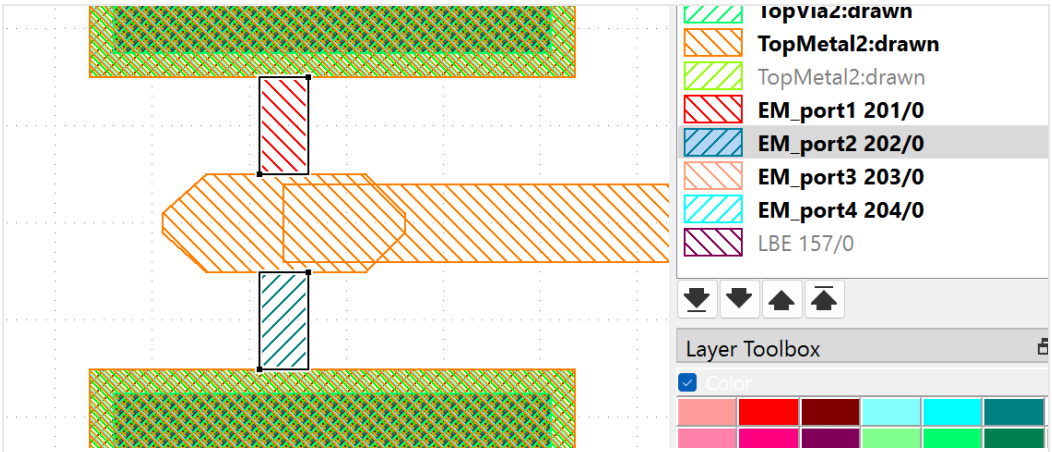



*Figure 14: Port layout for GSG configuration, composite port from 2 port elements*

Combining the results from 4 openEMS ports to 2 GSG ports is done later in the code, after the simulation is finished. The related simulation model examples are run_line_GSG.py and run_line_GSG_complex.py

## Simulation section in the model code

Coming back to the 1-port inductor example, the next section in the model code can be used as-is, not need to modify that. The code reads GDSII and stackup file and prepares the simulation.

```
# get technology stackup data
materials_list, dielectrics_list, metals_list = stackup_reader.read_substrate (XML_filename)
# get list of layers from technology
layernumbers = metals_list.getlayernumbers()
layernumbers.extend(simulation_ports.portlayers)

# read geometries from GDSII, only purpose 0
allpolygons = gds_reader.read_gds(gds_filename, layernumbers, purposelist=[0], \
                                  metals_list=metals_list, preprocess=preprocess_gds)


# calculate maximum cellsize from wavelength in diecletric
wavelength_air = 3e8/fstop
max_cellsize = (wavelength_air/unit)/(sqrt(materials_list.eps_max)*cells_per_wavelength)
```

The next section where the user needs to configure something is here:

```
# Create simulation for port 1+2 excitation
# Excited GSG port on left side is composite from CSX ports 1+2, opposite polarity,
# so we excite 1+2 simultaneously
excite_ports = [1,2]  # list of ports that are excited for this simulation run

FDTD = openEMS(EndCriteria=exp(energy_limit/10 * log(10)))
FDTD.SetGaussExcite( (fstart+fstop)/2, (fstop-fstart)/2 )
FDTD.SetBoundaryCond( Boundaries )

# use polygon-based meshing
FDTD = simulation_setup.setupSimulation (excite_ports, simulation_ports, FDTD, materials_list, dielectrics_list, metals_list,\
                                         allpolygons, max_cellsize, refined_cellsize, margin, unit, \
                                         xy_mesh_function=util_meshlines.create_xy_mesh_from_polygons)
# run simulation
sub1_data_path = simulation_setup.runSimulation (excite_ports, FDTD, sim_path, model_basename, preview_only, postprocess_only)
```

This is where the actual simulation is done, and just before simulation**, it is defined what ports must be excited for this simulation run**. For our inductor model, that is port number 1 only.

If we would simulate a simple 2-port model with standard ports, and we want to calculate S11, S21 from port 1 excitation as well as S22,S12 from port 2 excitation, we need to excite both ports, one after another, and the code would look like this:

```
# Create simulation for port 1 and 2 excitation, return value is list of data paths, one for each excitation
data_paths = []
for excite_ports in [[1],[2]]:  # list of ports that are excited one after another
    FDTD = openEMS(EndCriteria=exp(energy_limit/10 * log(10)))
    FDTD.SetGaussExcite( (fstart+fstop)/2, (fstop-fstart)/2 )
    FDTD.SetBoundaryCond( Boundaries )
    FDTD = simulation_setup.setupSimulation (excite_ports, simulation_ports, FDTD, materials_list, dielectrics_list, metals_list, \
                                             allpolygons, max_cellsize, refined_cellsize, margin, unit, \
                                             xy_mesh_function=util_meshlines.create_xy_mesh_from_polygons)
    data_paths.append(simulation_setup.runSimulation (excite_ports, FDTD, sim_path, model_basename, preview_only, postprocess_only))
```

This would then give 2 results directories, one for each port excitation.

## Evaluation of FDTD simulation results

The final step is evaluation of the data, which can be rather different depending on the device under test.

The actual simulation that we start with simulation_setup.runSimulation () returns the data directory for that excitation, with all FDTD results. The data in that directory needs some processing before we actually get S-parameters.

```
########## evaluation of results with composite GSG ports ###########

if preview_only==False:

    print('Begin data evaluation')

    # define dB function for S-parameters
    def dB(value):
        return 20.0*np.log10(np.abs(value))

    # define phase function for S-parameters
    def phase(value):
        return angle(value, deg=True)

    f = np.linspace(fstart,fstop,numfreq)


    # get results, CSX port definition is read from simulation ports object
    # S12, S22 is available because we have simulated both port1 and port2  excitation
    s11 = utilities.calculate_Sij (1, 1, f, sim_path, simulation_ports)
    s21 = utilities.calculate_Sij (2, 1, f, sim_path, simulation_ports)
    s12 = utilities.calculate_Sij (1, 2, f, sim_path, simulation_ports)
    s22 = utilities.calculate_Sij (2, 2, f, sim_path, simulation_ports)

    # write Touchstone S2P file
    s2p_name = os.path.join(sim_path, model_basename + '.s2p')
    utilities.write_snp (np.array([[s11, s21],[s12,s22]]),f, s2p_name)

    print('Starting plots')

    figure()
    plot(f/1e9, dB(s11), 'k-',  linewidth=2, label='S11 [dB]')
    plot(f/1e9, dB(s22), 'r--',  linewidth=2, label='S22 [dB]')
    grid()
    legend()
    xlabel('Frequency (GHz)')
```

The *.s1p file is created in the simulation data directory, which is created as a new directory where the Python simulation model file is located.

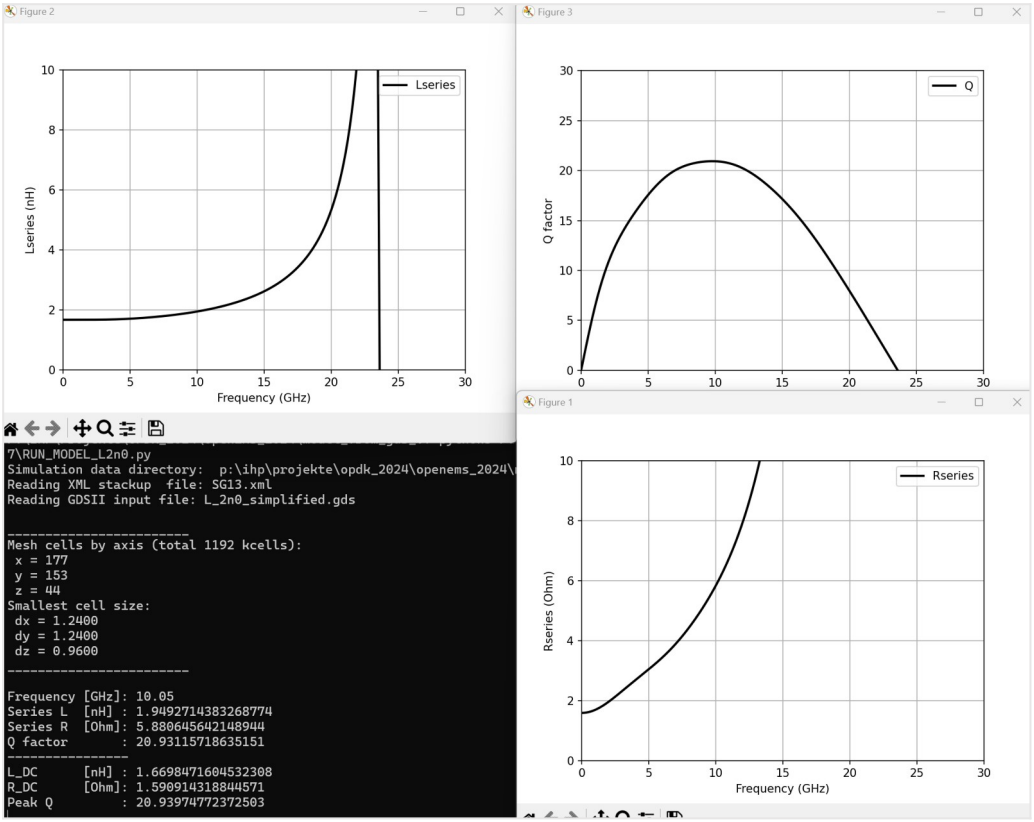



*Figure 15: Simulation results for inductor example*

**Please don't forget: To repeat evaluation of existing results, you can always go to the workflow settings at the beginning of the model file and change them to *preview_only=False* and *postprocess_only=True*, and then re-run the model to repeat evaluation and plots based on existing data. Change the code for evaluation if necessary, without repeating the full EM simulation.**

In the supplied examples, there is also an example for mixed port impedances, and an example for evaluation of GSG composite ports.

## GSG port evaluation of S-parameters

For a GSG port, we need to combine results of two openEMS port into one “composite” GSG port, as shown in the GSG model examples. Below is the code to calculate S11 and S21 for excitation from one side.

```
########## evaluation of results with composite GSG ports ###########

if not preview_only:

    # get results, CSX port definition is read from simulation ports object
    # left side CPW port consists of CSX ports 1 and 2
    CSX_port1 = simulation_ports.get_port_by_number(1).CSXport
    CSX_port2 = simulation_ports.get_port_by_number(2).CSXport
    # right side CPW port consists of CSX ports 3 and 4
    CSX_port3 = simulation_ports.get_port_by_number(3).CSXport
    CSX_port4 = simulation_ports.get_port_by_number(4).CSXport

    # S-parameters must combine results from both 100 Ohm CSX ports into one 50 Ohm GSG port
    f = np.linspace(fstart,fstop,numfreq)
    CSX_port1.CalcPort(sub1_data_path, f, 2*Z0)
    CSX_port2.CalcPort(sub1_data_path, f, 2*Z0)
    CSX_port3.CalcPort(sub1_data_path, f, 2*Z0)
    CSX_port4.CalcPort(sub1_data_path, f, 2*Z0)

    s11 = (CSX_port1.uf_ref + CSX_port2.uf_ref) / (CSX_port1.uf_inc + CSX_port2.uf_inc)
    s21 = (CSX_port3.uf_ref + CSX_port4.uf_ref) / (CSX_port1.uf_inc + CSX_port2.uf_inc)

    Zin = 0.5 * (CSX_port1.uf_tot + CSX_port2.uf_tot)  / (CSX_port1.if_tot + CSX_port2.if_tot)

    # S12, S22 is NOT available because we have NOT simulated port2 excitation
    # fake it by assuming symmetry
    s22 = s11
    s12 = s21

    # write Touchstone S2P file
    s2p_name = os.path.join(sim_path, model_basename + '.s2p')
    utilities.write_snp (np.array([[s11, s21],[s12,s22]]),f, s2p_name)
```

The reverse path for S22 and S12 is not simulated here, because the geometry is symmetric anyway. To simulate that, we would need two separate simulation runs with excitation from left and right side, resulting in two output data directories that can be evaluated using the same scheme.

## Choosing the meshing method

Two meshing methods are available so far. The default is equal mesh spacing across the entire area where GDSII elements are drawn. As an option, an automatic meshing algorithm will be used, which tries to detect edges and diagonal areas that need local refinement.

In both cases, the **refined_cellsize** value in the simulation settings section is used to define the minimum mesh cell size in the x-y plane. This determines the minimum geometry resolution, but also the resolution for local field distribution like skin effect in conductors.

**Note that the smallest cell size in the model determines the time step used in simulation, and smaller timesteps mean longer simulation time.**

To support you with detecting unwanted small mesh cells, the smallest mesh cell in each direction is printed when you run the model:

```
Reading XML stackup  file: SG13G2.xml
Reading GDSII input file: L_2n0_twoport.gds


________________________
Mesh cells by axis (total 3363 kcells):
 x = 267
 y = 229
 z = 55
Smallest cell size:
 dx = 0.8809
 dy = 0.8134
 dz = 0.4000
________________________

Starting AppCSXCAD 3D viewer with file:
 d:\perforce\volker_omen15_7389\volker_omen15_7389\openems_p
inductor_2port.xml
Close AppCSXCAD to continue or press <Ctrl>-C to abort
```

The vertical meshing in z-direction is created automatically based on the XML stackup file and the layers used in the GDSII file.

At the beginning of each simulation run, or when running with *preview_only=True* in the workflow settings, the AppCSXCAD 3D viewer is started where you can check the mesh in detail. The smallest mesh cell in x-y plane should be no less than your setting for *refined_cellsize* in the simulation settings, or the smallest mesh cell resulting from the stackup, whichever is smaller.

For reference: the thickness of the bottom metal layers in stackup is 0.42µm and the thickness of passivation is 0.4µm. They often determine the smallest mesh dimension overall, unless you have specified a smaller value of the refined_cellsize variable.

## Enable automatic meshing

By default, constant distance of mesh lines is used for the region covered by GDSII polygons, with graded mesh lines outside.

Automatic meshing is enabled if the model line for simulation setup

FDTD = simulation_setup.setupSimulation (…)

includes the optional parameter

xy_mesh_function=util_meshlines.create_xy_mesh_from_polygons

at the end of the line.

```
# Create simulation for port 1+2 excitation
# Excited GSG port on left side is composite from CSX ports 1+2, opposite polarity,
# so we excite 1+2 simultaneously
excite_ports = [1,2]  # list of ports that are excited for this simulation run

FDTD = openEMS(EndCriteria=exp(energy_limit/10 * log(10)))
FDTD.SetGaussExcite( (fstart+fstop)/2, (fstop-fstart)/2 )
FDTD.SetBoundaryCond( Boundaries )

# use polygon-based meshing
FDTD = simulation_setup.setupSimulation (excite_ports, simulation_ports, FDTD, materials_list, dielectrics_list, metals_list,\
                                         allpolygons, max_cellsize, refined_cellsize, margin, unit, \
                                         xy_mesh_function=util_meshlines.create_xy_mesh_from_polygons)
# run simulation
sub1_data_path = simulation_setup.runSimulation (excite_ports, FDTD, sim_path, model_basename, preview_only, postprocess_only)
```

Without that optional parameter, equal meshing is used. In the future, additional mesh algorithms might be added, which can then be called using this xy_mesh_function parameter.

# Antenna simulation

Antenna simulation is special because it requires additional spacing between geometries+stackup and the simulation boundary, with PML_8 perfectly matched layers specified for absorbing simulation boundaries. In the simulation model, we usually want to add field recording, so that we can calculate the antenna pattern.

The example below shows these settings that are different from other models:

## Absorbing boundaries

The radiation from the antenna must be absorbed at the simulation boundaries, and PML_8 is the best boundary condition for that. The PML_8 boundary adds 8 layers of absorbing material on each side, which fills 8 mesh cells inside the simulation area.

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

unit   = 1e-6   # geometry is in microns
margin = 100    # distance in microns from GDSII geometry boundary to chip boundary

fstart = 200e9
fstop  = 300e9
numfreq = 401

ftarget = 245e9  # frequency for antenna pattern calculation

refined_cellsize = 2.5  # mesh cell size in conductor region

# choices for boundary:
# 'PEC' : perfect electric conductor (default)
# 'PMC' : perfect magnetic conductor, useful for symmetries
# 'MUR' : simple MUR absorbing boundary conditions
# 'PML_8' : PML absorbing boundary conditions
Boundaries = ['PML_8', 'PML_8', 'PML_8', 'PML_8', 'PML_8', 'PML_8']

cells_per_wavelength = 12   # how many mesh cells per wavelength, must be 10 or more
energy_limit = -40          # end criteria for residual energy (dB)

# ports from GDSII Data, polygon geometry from specified special layer
# note that for multiport simulation, excitations are switched on/off in simulation_setup.createSimulation below
simulation_ports = simulation_setup.all_simulation_ports()
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1, voltage=1, port_Z0=50, source_layernum=201, \
                                                           target_layername='TopMetal2', direction='x'))
```

The energy limit (convergence criteria) is reduced to -40dB here, which is enough for reasonable accuracy for return loss and antenna pattern. This converges faster than the -50dB that we used for the inductor example, where Q factor calculation requires higher accuracy.

## Simulation setup

In the simulation setup for this antenna, the optional parameter *air_around* adds additional air spacing all around the model, to ensure proper distance from the absorbing boundaries.

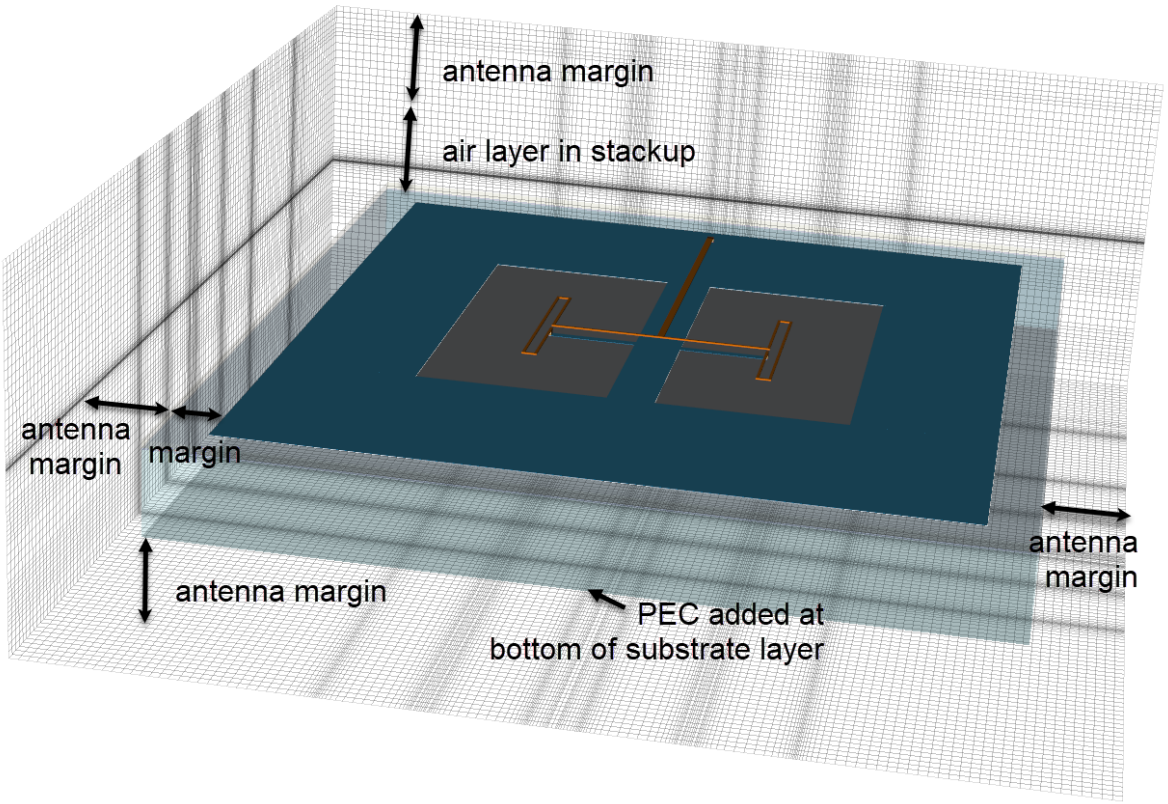



*Figure 16: Effect of is_antenna=True option*

Another difference from the inductor example is that a **nf2ff_box** is added after the call to *simulation_setup.setupSimulation()* to do the field sampling required for antenna pattern calculation.

```
# Prepare simulation for port 1 excitation
excite_ports = [1]  # list of ports that are excited for this simulation run
FDTD = openEMS(EndCriteria=exp(energy_limit/10 * log(10)))
FDTD.SetGaussExcite( (fstart+fstop)/2, (fstop-fstart)/2 )
FDTD.SetBoundaryCond( Boundaries )

FDTD = simulation_setup.setupSimulation (excite_ports, simulation_ports, FDTD, materials_list, dielectrics_list, metals_list, \
                                         allpolygons, max_cellsize, refined_cellsize, margin, unit, \
                                         xy_mesh_function=util_meshlines.create_xy_mesh_from_polygons,\
                                         air_around=0.5*wavelength_air/unit)

# add nf2ff box for antenna pattern calculation
nf2ff_box = FDTD.CreateNF2FFBox(opt_resolution=[max_cellsize]*3,  frequency = [ftarget])

# run simulation
sub1_data_path = simulation_setup.runSimulation (excite_ports, FDTD, sim_path, model_basename, preview_only, postprocess_only)
```

The call to function CreateNF2FFBox() is plain openEMS syntax, so check out the documentation available there:
https://docs.openems.de/python/openEMS/nf2ff.html
https://github.com/thliebig/openEMS/blob/master/python/openEMS/nf2ff.py.

Parameter frequency is optional, to store only the field data for that one frequency, which saves disk space. Without this optional parameter, nf2ff stores all time domain data, and patterns can be calculated in postprocessing for any frequency. The disadvantage is that data files will be much larger then.

For the evaluation the NF2FF data files and creating the antenna pattern plot, we also use plain openEMS syntax. Have a look at example RUN_MODEL_DUAL_DIPOLE.py for more details.

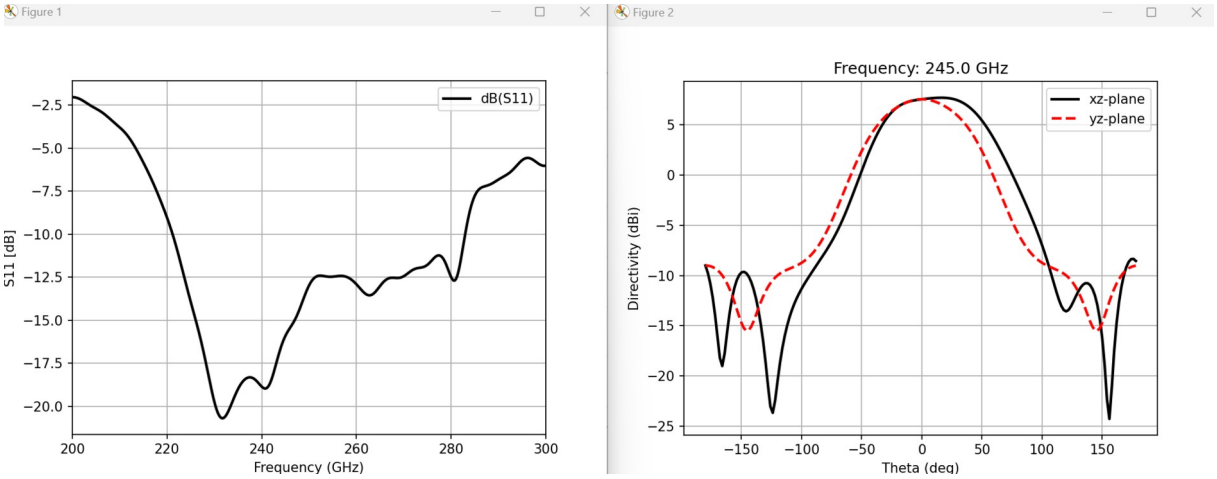



*Figure 17: Return loss and directivity*

# FAQ

## How do I use the example models?

Open a command window in the directory where the model files are located.
Examples are identified by a name starting with run_...

Before you start simulating the model, open the model *.py file in a text editor and check the values of *preview_only* and *postprocess_only*.

| **preview_only** | **postprocess_only** | **Action** |
|---|---|---|
| False | False | 3D model preview, simulation starts immediately when 3D viewer is closed, postprocess & plot |
| True | False | 3D model preview only (to check model visually during model development) |
| False | True | Postprocess and plot results from existing data (simulation done before) |

To start the action, type p*ython3* followed by the name of the run_... filename

```
python3 run_dual_dipole.py
```

Note that running the actual simulation will take some time, so you better pre-check the model carefully in the AppCSXCAD 3D viewer.

## How to create my own model?

The best method is to start from the example files that matches best your simulation type.

| | |
|---|---|
| run_inductor_diffport.py | Inductor with 1 port across terminals, plot L and Q, create *.s1p file |
| run_inductor_diffport.py | Inductors with 2 ports to a “faked” common ground reference at the surface of the substrate, see comments in source code |
| run_transformer_diffport.py | Transformer with 2 ports, mixed impedance levels, via array merging enabled. |
| run_line_GSG.py | Line with two GSG port, each built from 2 EM ports |
| run_line_GSG_complex.py | Much like run_line_GSG.py, but more complex layout that requires settings for GDSII pre-processing. |
| run_line_viaport.py | Microstrip with via ports between Metal1 and TopMetal2, to show how these ports across many layers are defined. The model uses a stackup with no substrate below, because that is shielded by the Metal1 plane anyway. |
| run_line_full2port.py | Line that is excited from both ends, to get full S-parameter in both directions. Most other examples simulate one signal path only, giving only S11 and S21, and fake the reverse path by using symmetry |
| run_dual_dipole.py | Antenna example using LBE cutout, shows how to add air around the entire model for absorbing boundaries, shows how to add the field sampling for antenna pattern. |
| run_rfcmim_2port_full.py | MIM with via array merging of GDSII data. MIM dielectric is faked in stackup, to avoid ultra thin layer that would require ultra small timestep |

## What is a good mesh size?

This is not easy to answer – finer mesh is closer to reality, but simulation takes much longer. One obvious limitation is fine geometry detail, the mesh must resolve all relevant detail. Also, we discretize into skin effect and mesh size needs to capture surface currents and skin effect.

Depending on frequency, something like 1 to 0.5 (microns) is a good starting point. When you are new to openEMS, you might want to run different mesh density and compare results.

## Why is my simulation slow?

Simulation time depends on the number of mesh cells and the simulation timestep, which is calculated from the smallest mesh cell in your model. When the model is meshed, the code will display a mesh summary with total mesh count and smallest cell in x,y and z direction.

The smallest mesh cell in x,y direction results from the *refined_cellsize* value in your model file.

The smallest mesh cell in z direction usually results from the passivation thickness (0.4µm) or the lower metal layers (0.42µm).

To reduce the total mesh count, without compromising the mesh resolution for fine detail, you might try the automatic meshing. This tries to create an efficient smart mesh, with high resolution only where it is needed. See chapter “Enable automatic meshing” for more details.

The default uniform mesh uses constant mesh size of *refined_cellsize* across the entire GDSII drawing area, resulting in larger total mesh count.

## Can I reduce the stackup by removing the substrate for transmission line models?

This is not implemented yet, because a PEC bottom boundary is lossless. If we use this as the ground plane for transmission line models, all the loss in that lower metal plane is removed.

In IHP SG13G2, the lower metals are rather thin, so results for loss will not be accurate with a PEC boundary as the return conductor.

## Can I use a different data directory location?

This is defined in your model code. In the examples, the data directory is created below the model directory, but you can change that in your model code.

## My imported GDSII looks strange

Some complex shapes with many cutouts might require additional preprocessing, which can be enabled in the model file:

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

Also see example RUN_MODEL_GSG2.py where this pre-processing is required.

## My results show a lot of ripple

This might happen if simulation stops when the energy in the simulation volume is still high. Check your energy limit in the model file: -40dB is a good default but for precise Q factor it can be -50dB or even lower.

```
energy_limit = -40          # end criteria for residual energy (dB)
```

## Where do I change the view style in AppCSXCAD 3D viewer?

To see a 2D view with no perspective distortion, use the 2D view mode, and set the axis accordingly. You can also move the displayed mesh plane position: it be moved in z direction to top, so that the mesh lines are not hidden by metals.

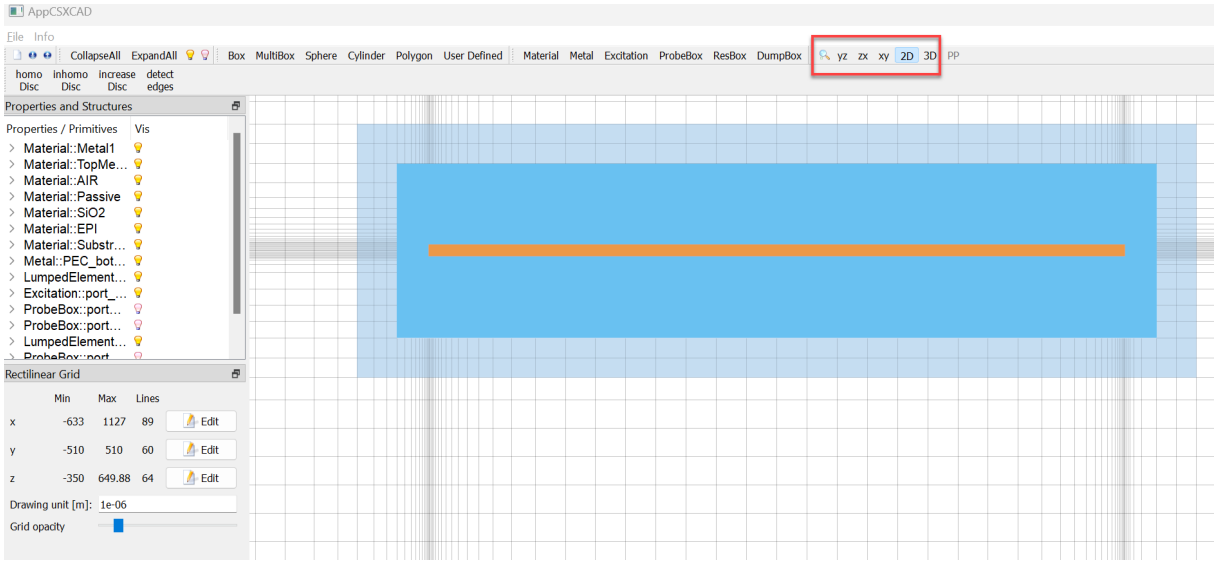



*Figure 18: AppCSXCAD view*

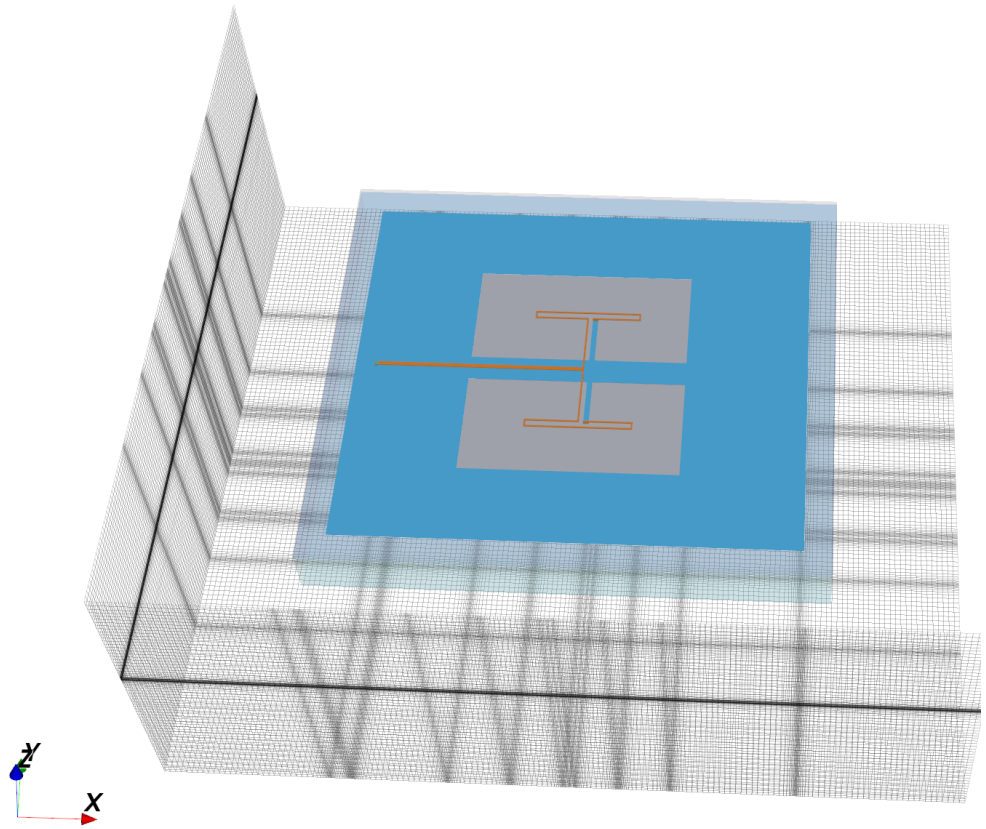
## I spotted a model mistake in 3D viewer, can I quit without simulating?

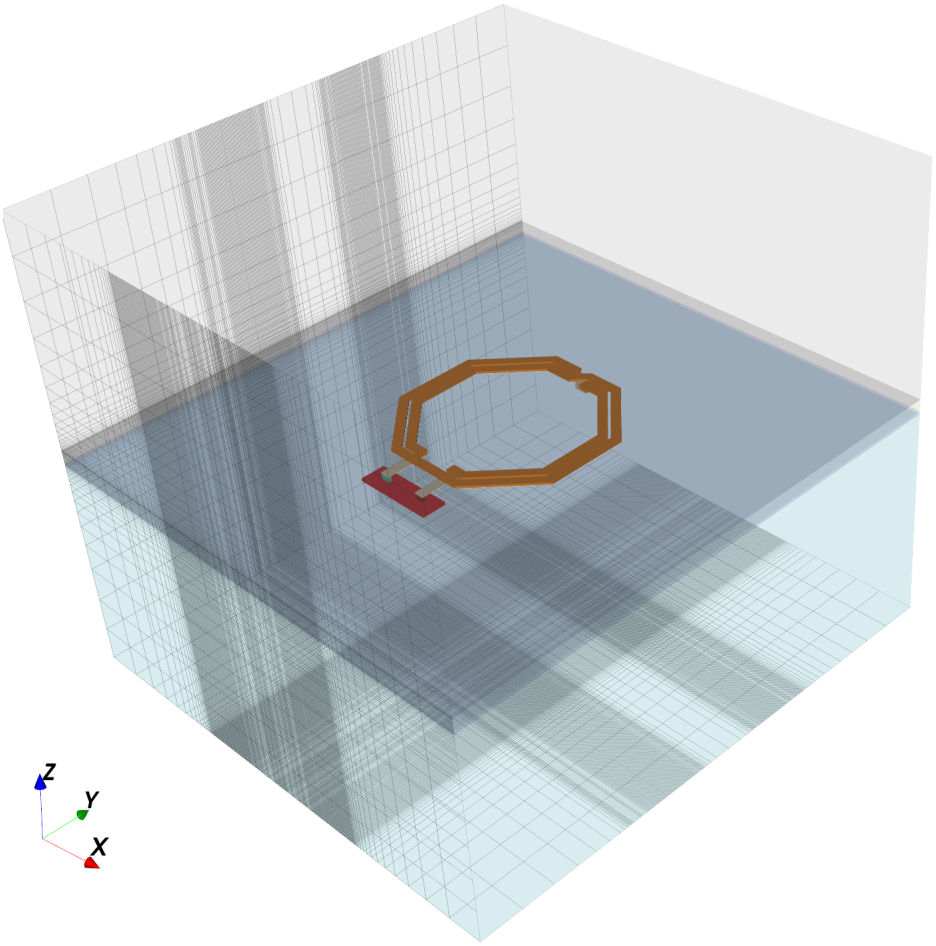
If you are running with *preview_only=True*, you can simply close the viewer and that's it.

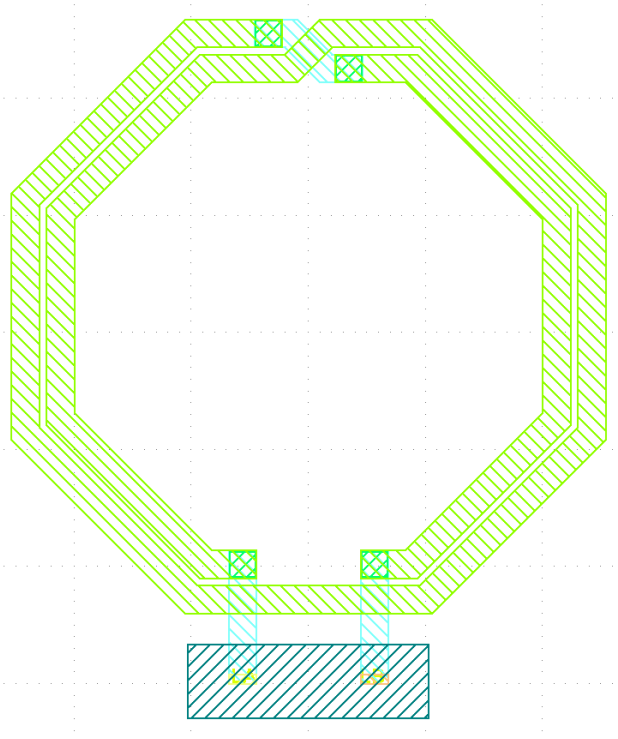
If you configured a full simulation run using *preview_only=False*, closing the viewer will start simulation. However, you can exit by pressing Ctrl-C while the 3D viewer is still open.

## List of example models

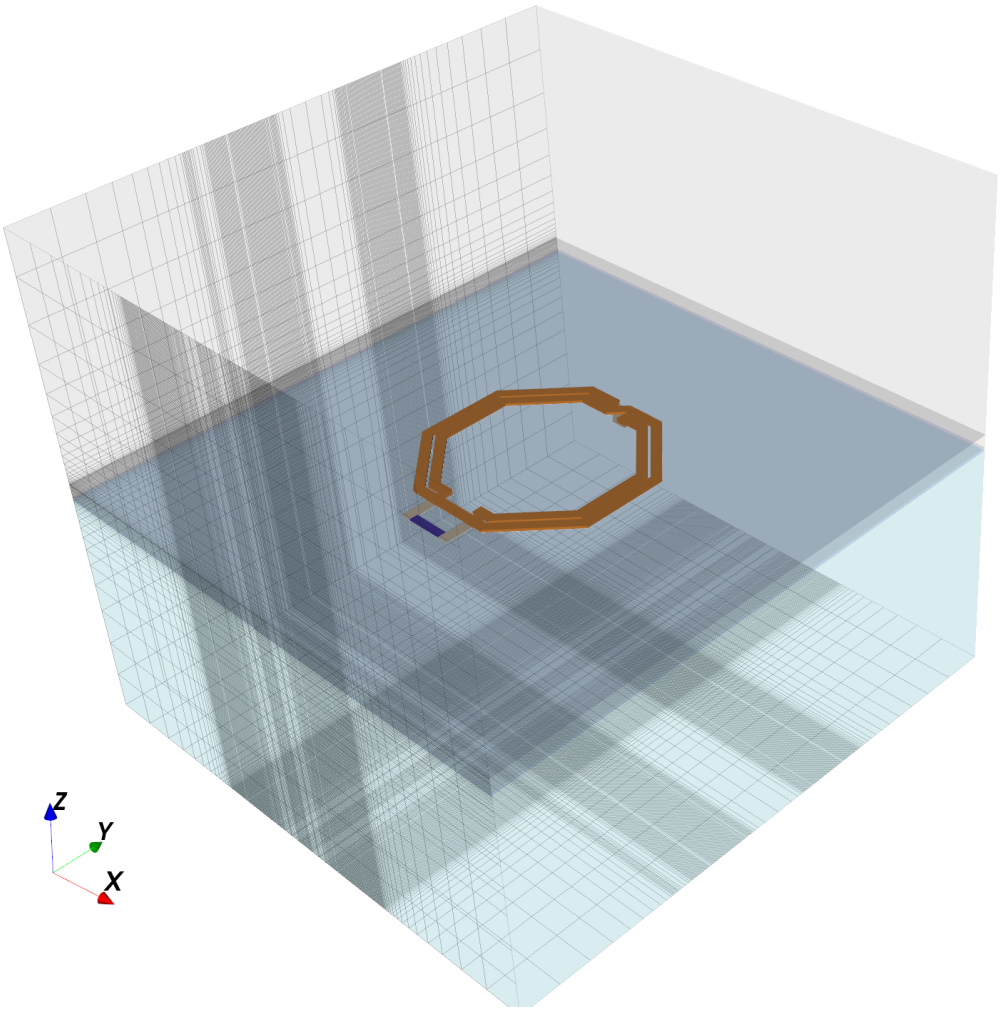
run_dual_dipole.py

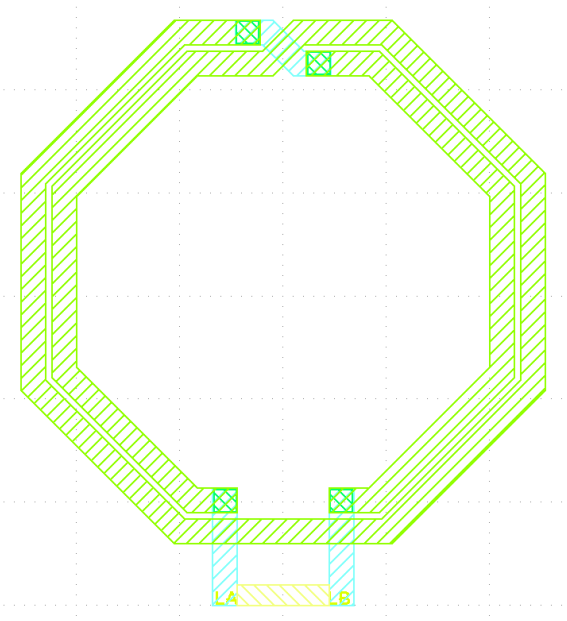



z
y
x

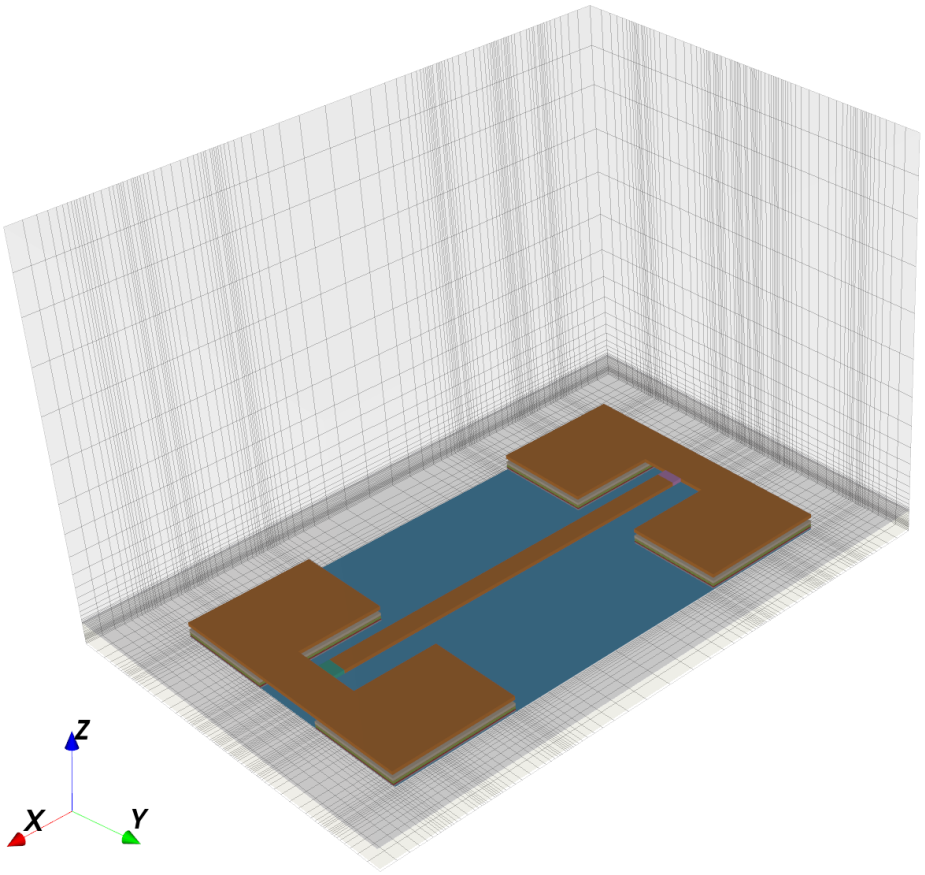


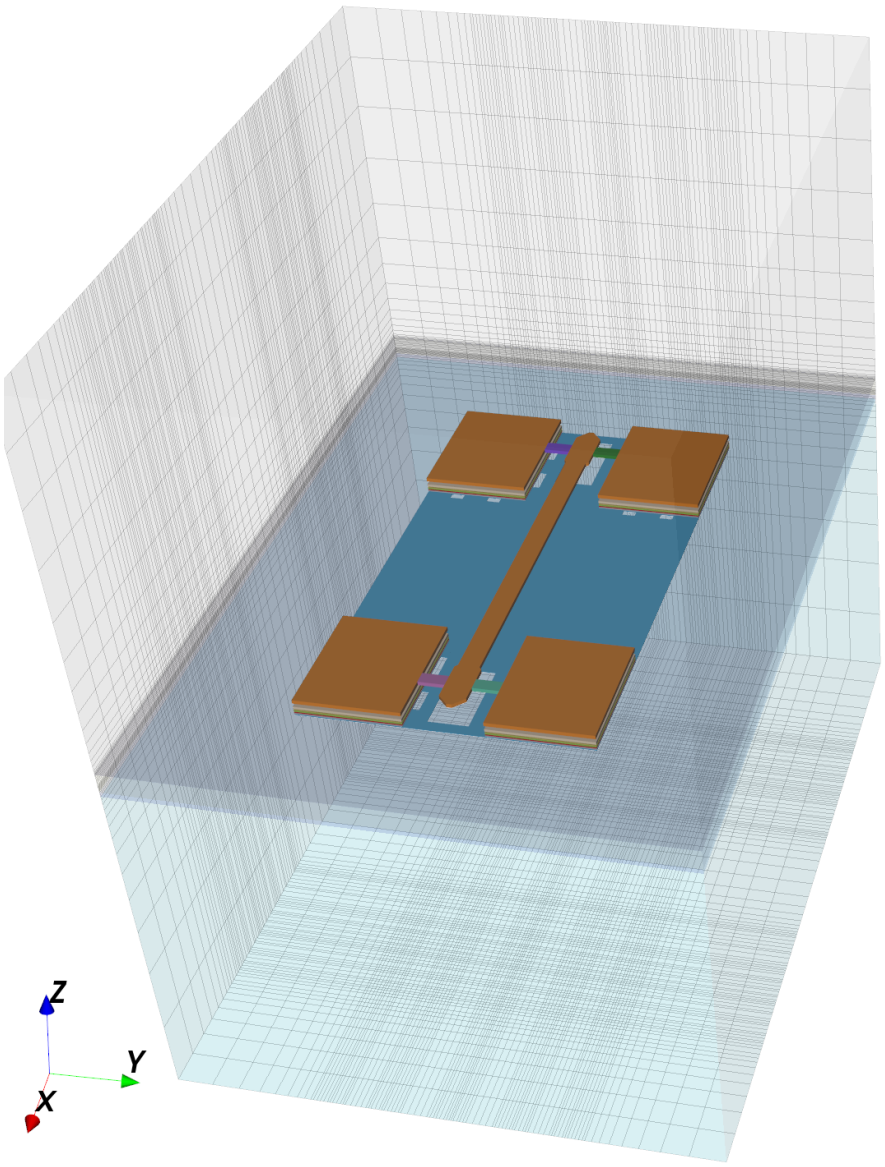
run_inductor_diffport.py

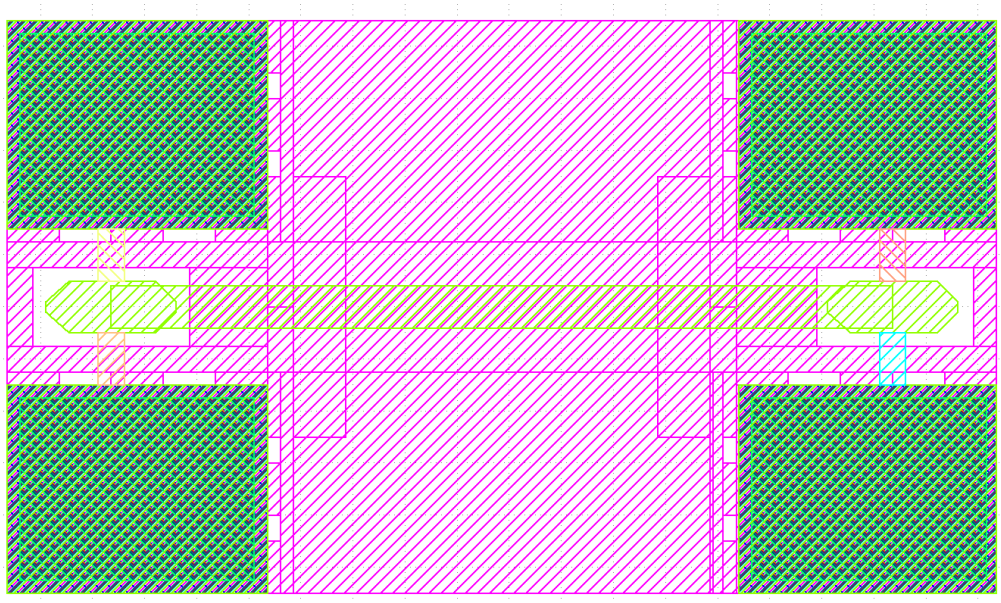


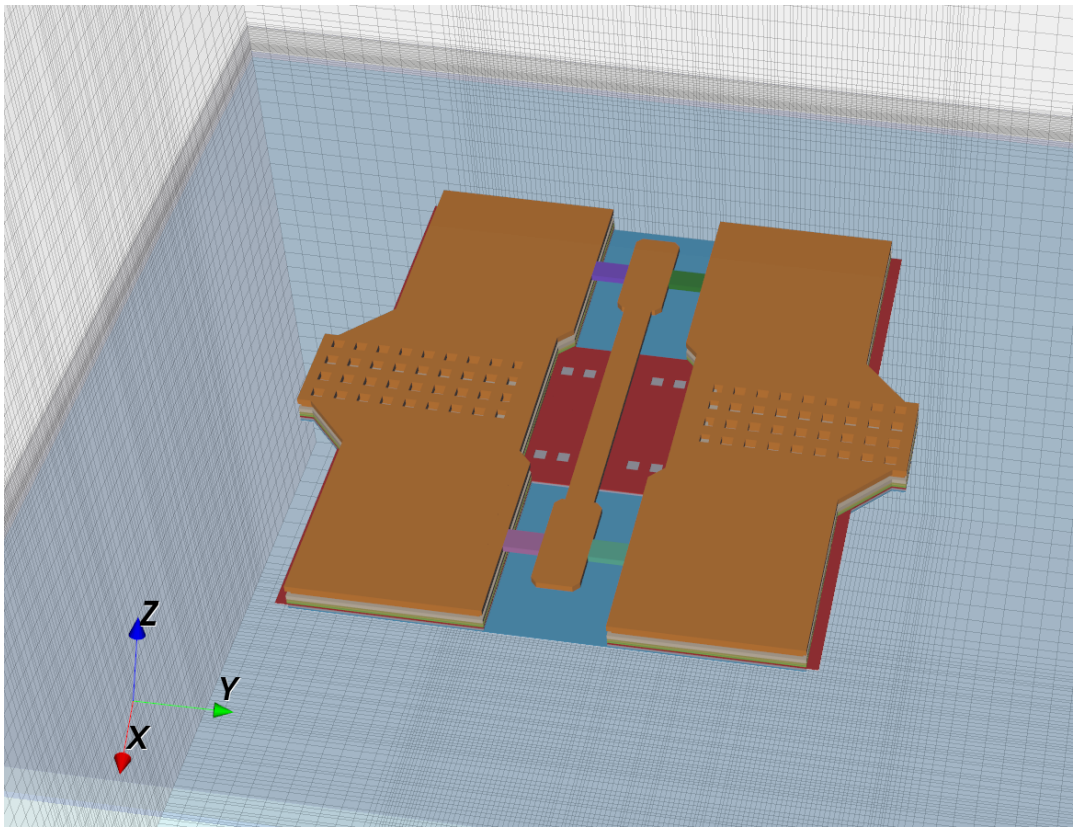
run_line_full2port.py

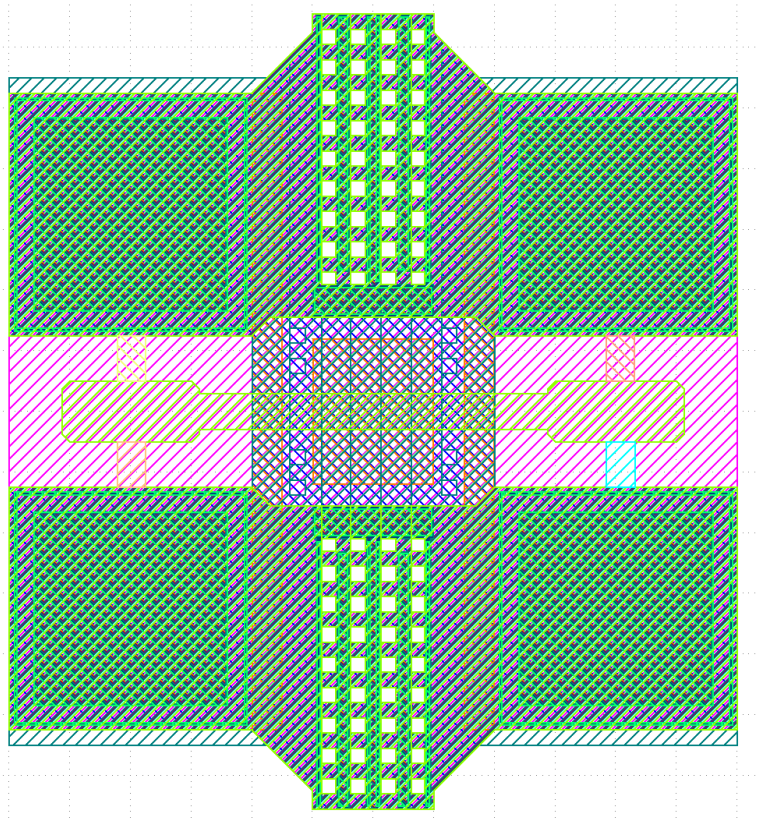



Z
Y
X

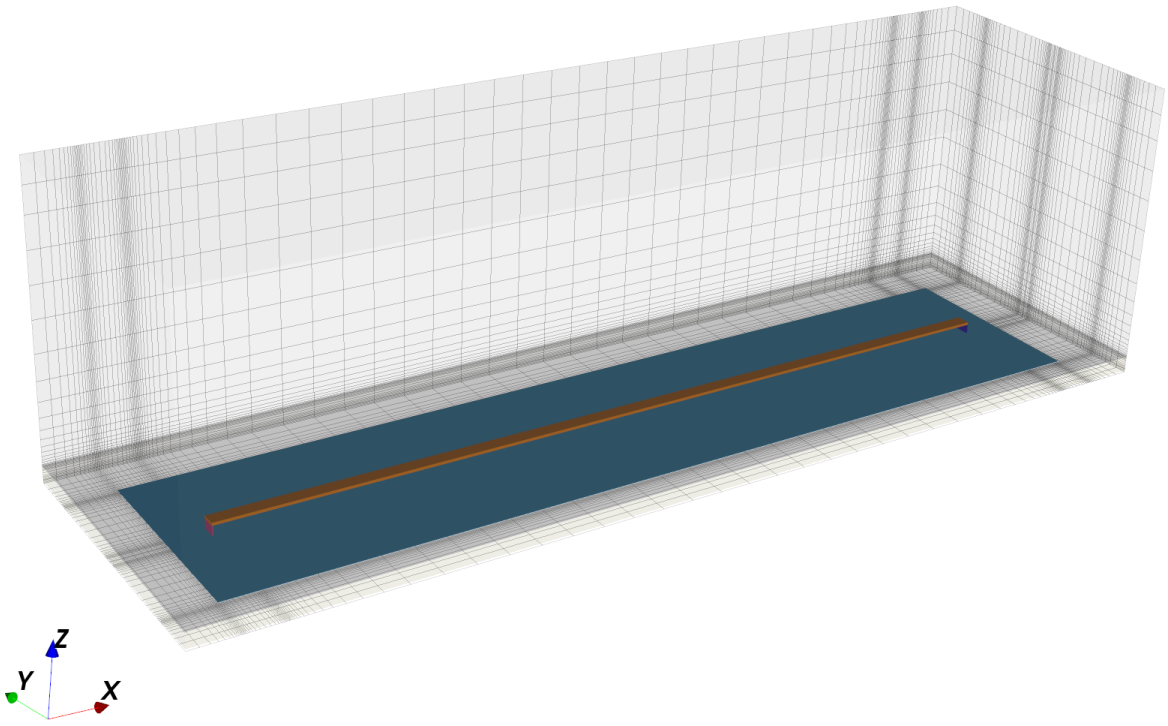


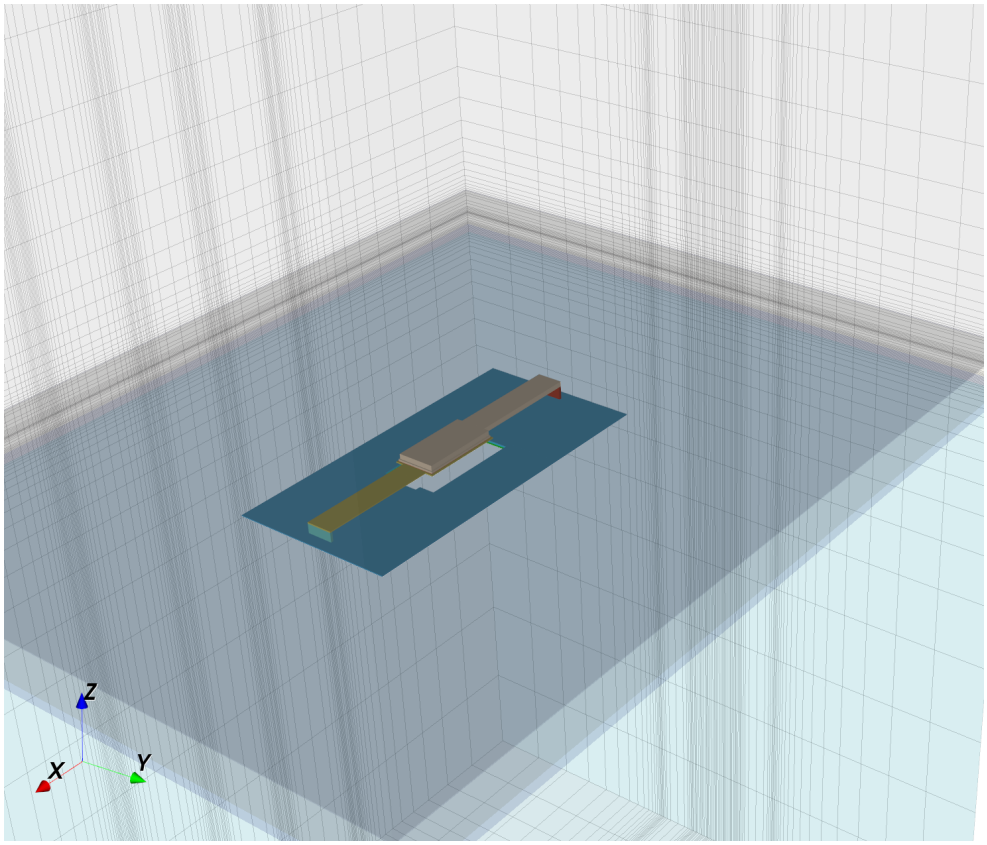
run_line_GSG_complex.py

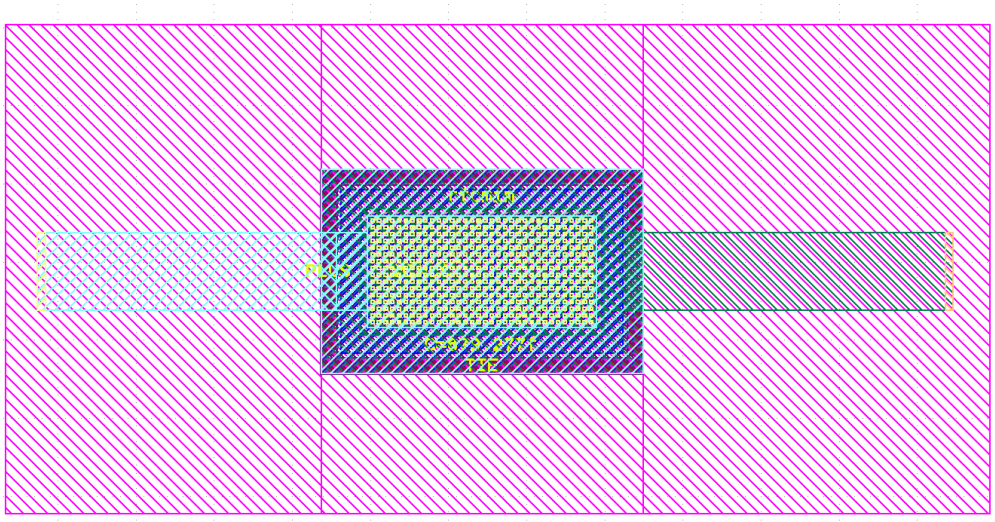


run_line_viaport.py




Z
X
Y



run_transformer_diffport.py