User-friendly FDTD EM Workflow for IHP OpenPDK with Automatic Meshing

Volker Muehlhaus, Dr. Muehlhaus Consulting & Software GmbH

ESS ERC Workshop W2 on September 8, 2025

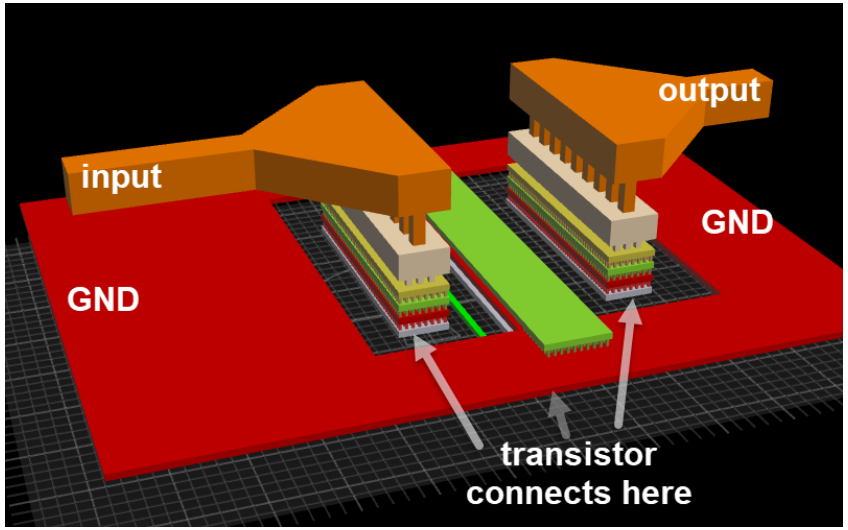
Leibniz Institute
for High
Performance
Microelectronics

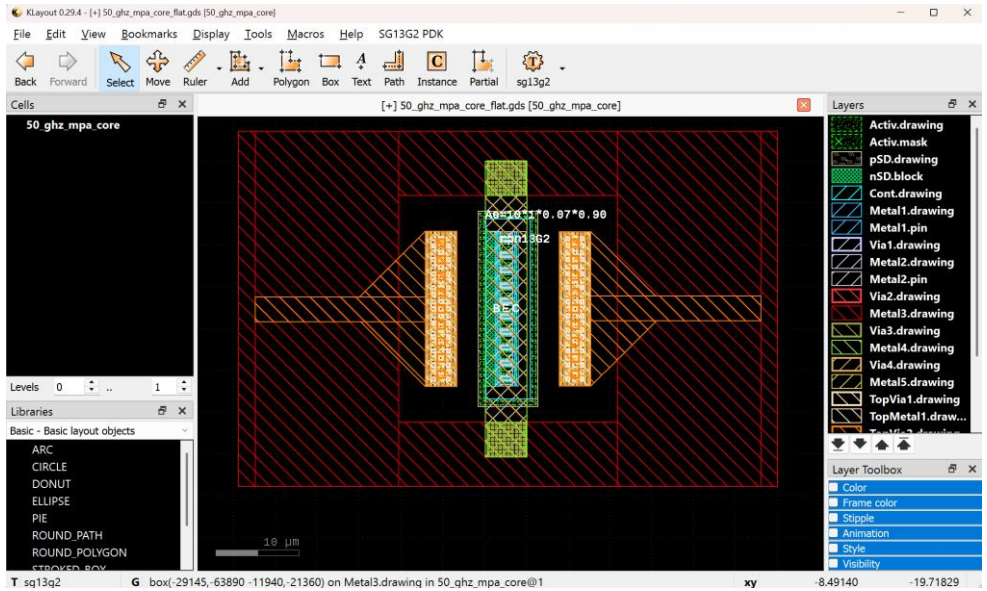
Overview

1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Typical simulation run
5. Automatic Meshing Algorithm
6. Simulated vs. measured
7. Summary and Outlook

Why use Electromagnetic (EM) simulation?

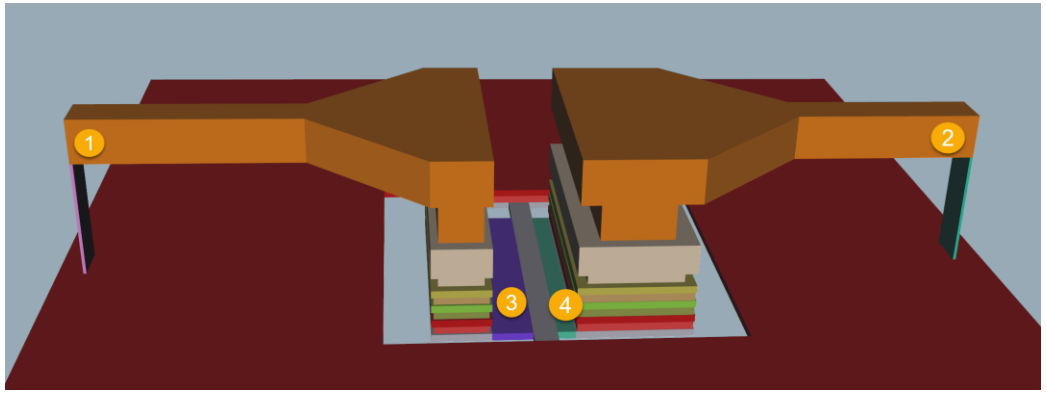
- Interconnect layout adds series R,L and shunt C
- No electrical model of layout/interconnect in the PDK
- We can use EM simulation for this part of design
- FDTD method in openEMS valid up to highest frequencies

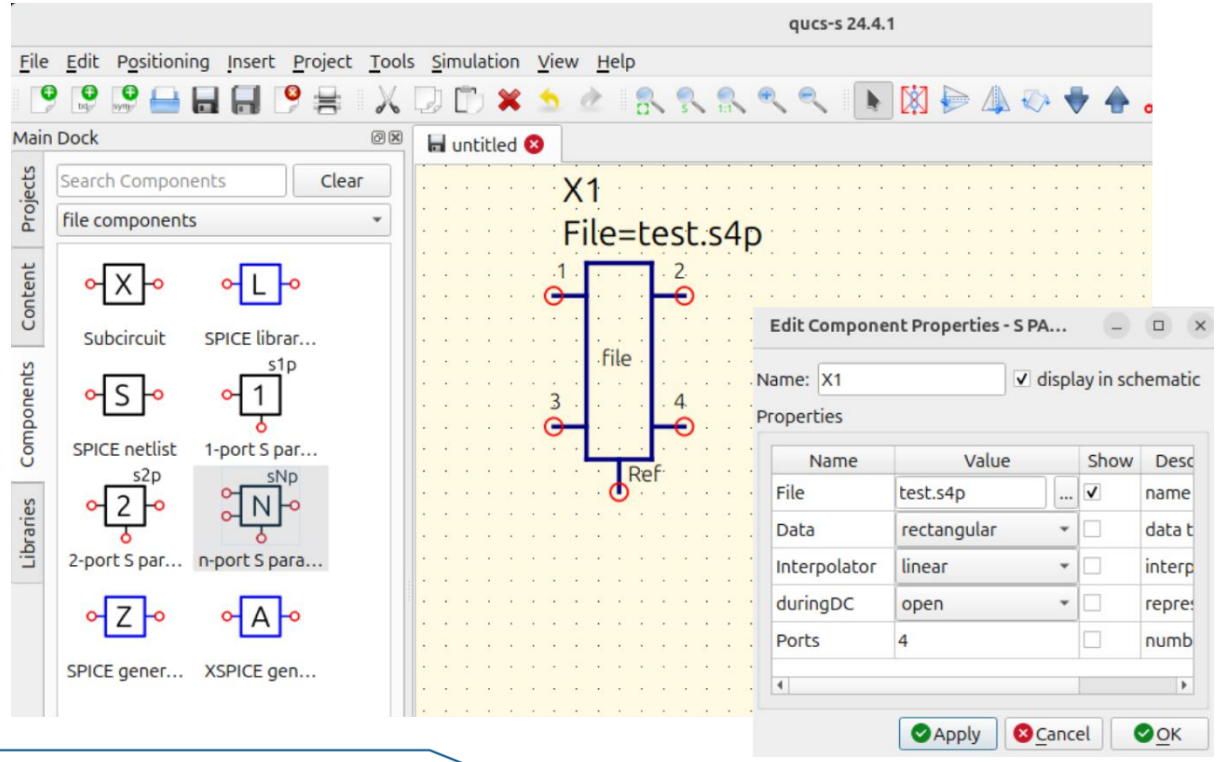




EM simulation in the overall workflow

- Need to separate layout/routing from device models, EM model includes layout only
- Create “ports” at the interfaces where EM model connects to circuit models
- Run in EM simulator, result is black box data over frequency
- Connect EM Black box data with device models in circuit simulation

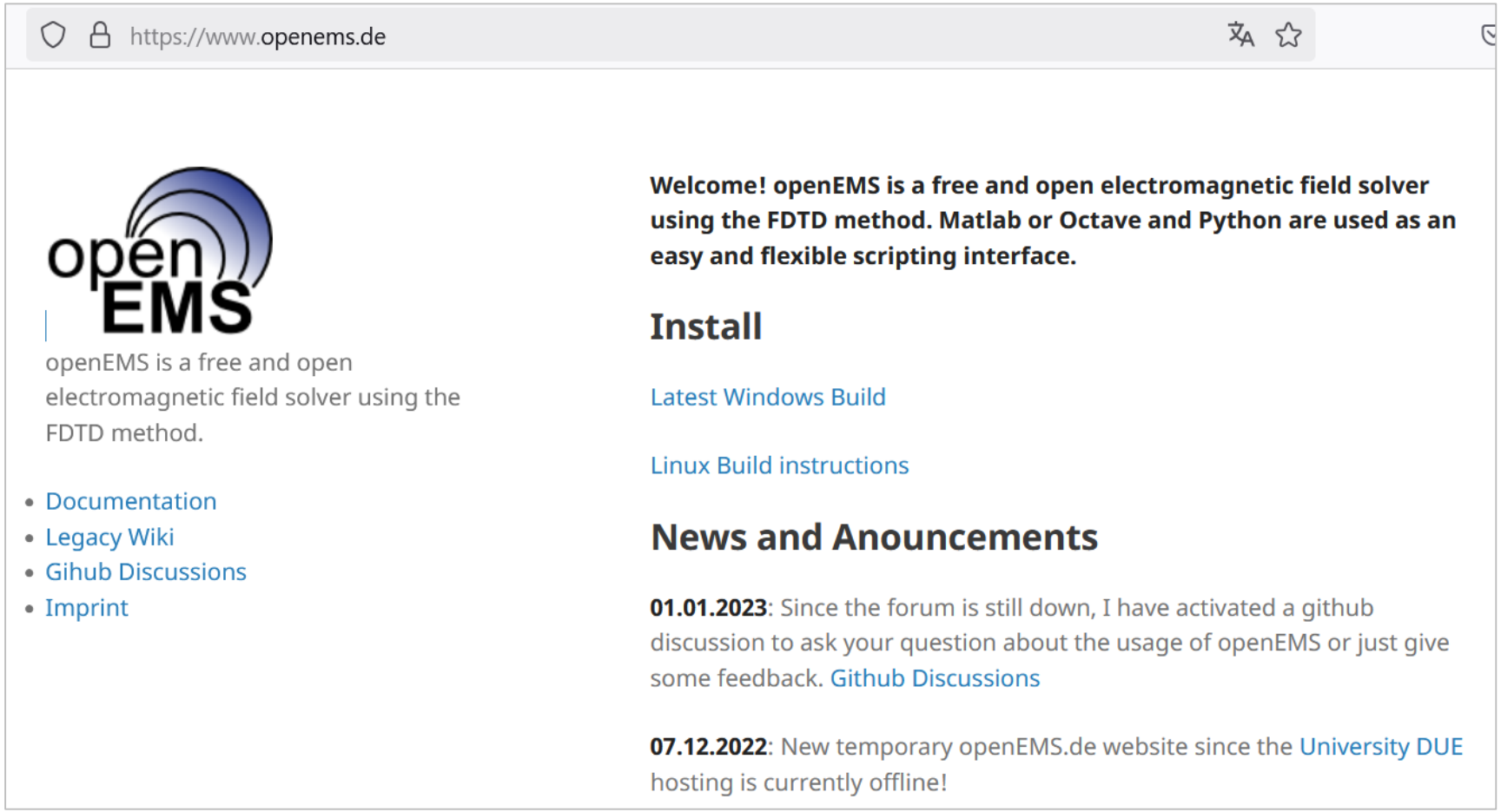




Overview

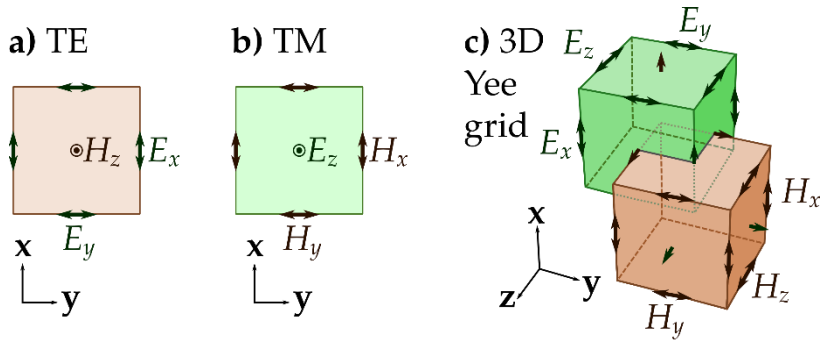
1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Typical simulation run
5. Automatic Meshing Algorithm
6. Simulated vs. measured
7. Summary

Basis of this work: openEMS by Thorsten Liebig

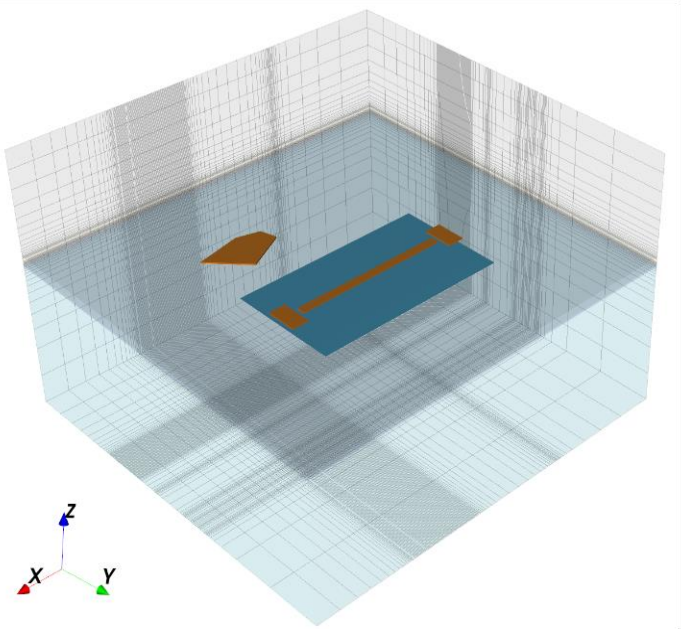


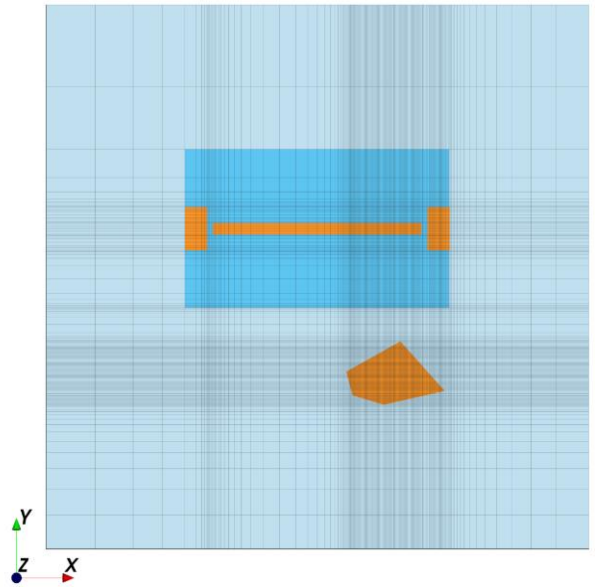
FDTD Method used in openEMS: Mesh

- Simulation method is **F**inite **D**ifference in **T**ime **D**omain
- Simulation domain is divided into many small boxes
- Calculates all E, H value for each box at one timestep, then next timestep, next timestep…



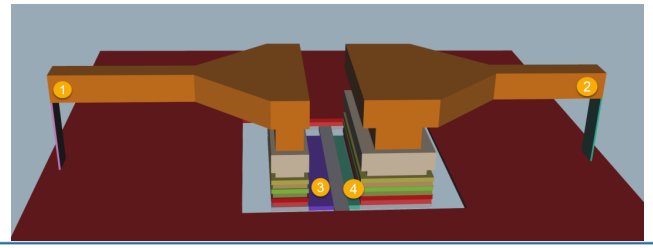
Von FDominec - Eigenes Werk, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=37637085

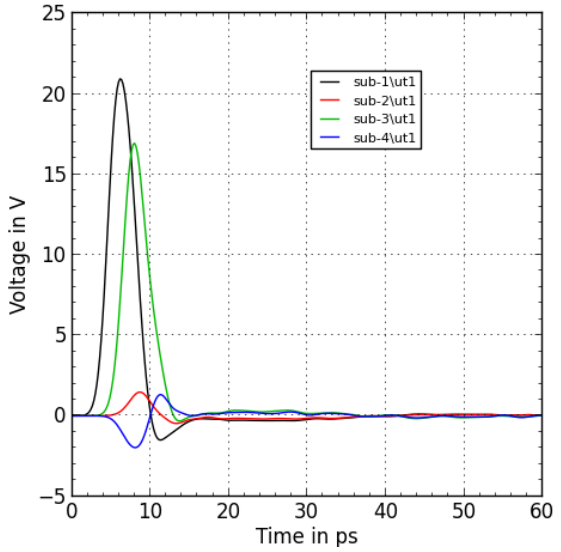




FDTD Method used in openEMS: Time Domain

- Excite one port with gaussian pulse => wideband continuous spectrum
- FDTD calculates E and H in time domain, step by step, until energy in model is (near) zero
- From time signals at ports, we get wideband S-parameters using Fourier transform (FFT)
- This gives one column of S-matrix (wideband) per port excitation
- Repeat for all ports to get full [S] matrix





FFT

$$\begin{bmatrix} S_{11} & S_{12} & S_{13} & S_{14} \\ S_{21} & S_{22} & S_{21} & S_{24} \\ S_{31} & S_{32} & S_{33} & S_{24} \\ S_{41} & S_{42} & S_{43} & S_{44} \end{bmatrix}$$

Overview

1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Typical simulation run
5. Automatic Meshing Algorithm
6. Simulated vs. measured
7. Summary

Old method: RFIC EM model using plain openEMS

- Old EM model uses one large file with Python code for openEMS API
- All technology materials + stackup defined in that model code
- Full layout must be coded in the model, polygon by polygon, point by point
- Meshing of simulation model must be coded by user specific for his geometry or use inefficient uniform mesh across entire model
- Much „infrastructure“ and „housekeeping“ code blows up the model file

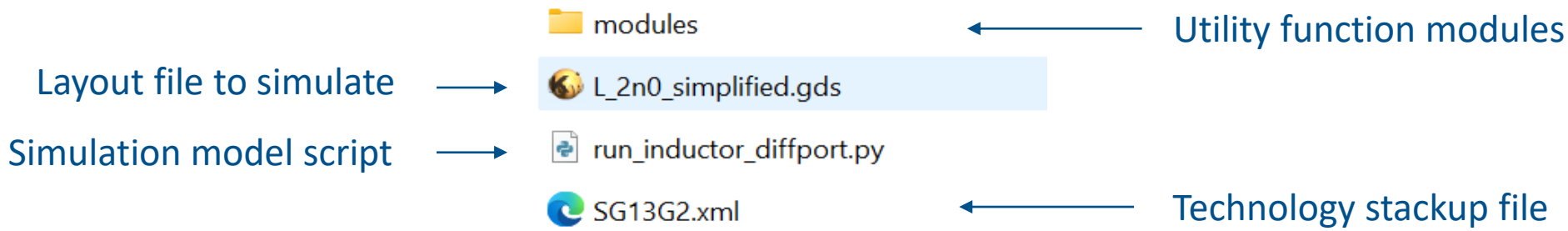
- **Large file, lots of code lines, difficult to create and difficult to re-use**
- **Slow simulation when using the simple uniform meshing**
- **We have a better solution now!**

New: Make openEMS easier to use for RFIC work

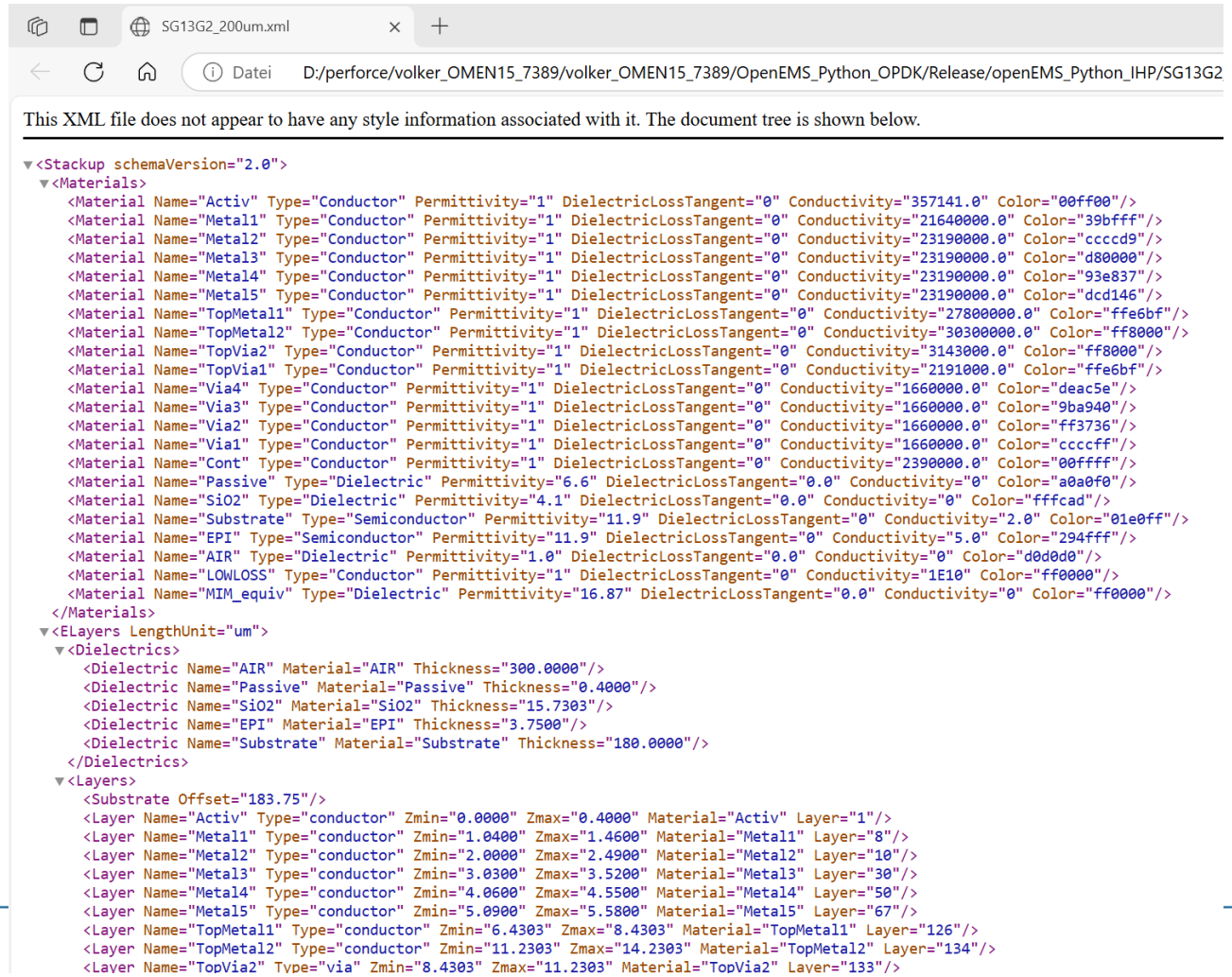
- https://github.com/VolkerMuehlhaus/openems_ihp_sg13g2
- New workflow adds **additional software layer** that specifically targets RFIC EM use cases
- Modular approach, keep the main model simple & clean, easy to re-use
- Read geometries directly from GDSII layout files (no prior conversion needed)
- Read all technology stackup & material data from XML file
- Geometry pre-processing options, e.g. via array merging
- Automatic meshing based on GDSII data, reduces total mesh count and speeds up simulation

- How to use: For new model, find the closest match from existing examples and change only a few code lines

New workflow minimum example



Geometry input file, stackup input file

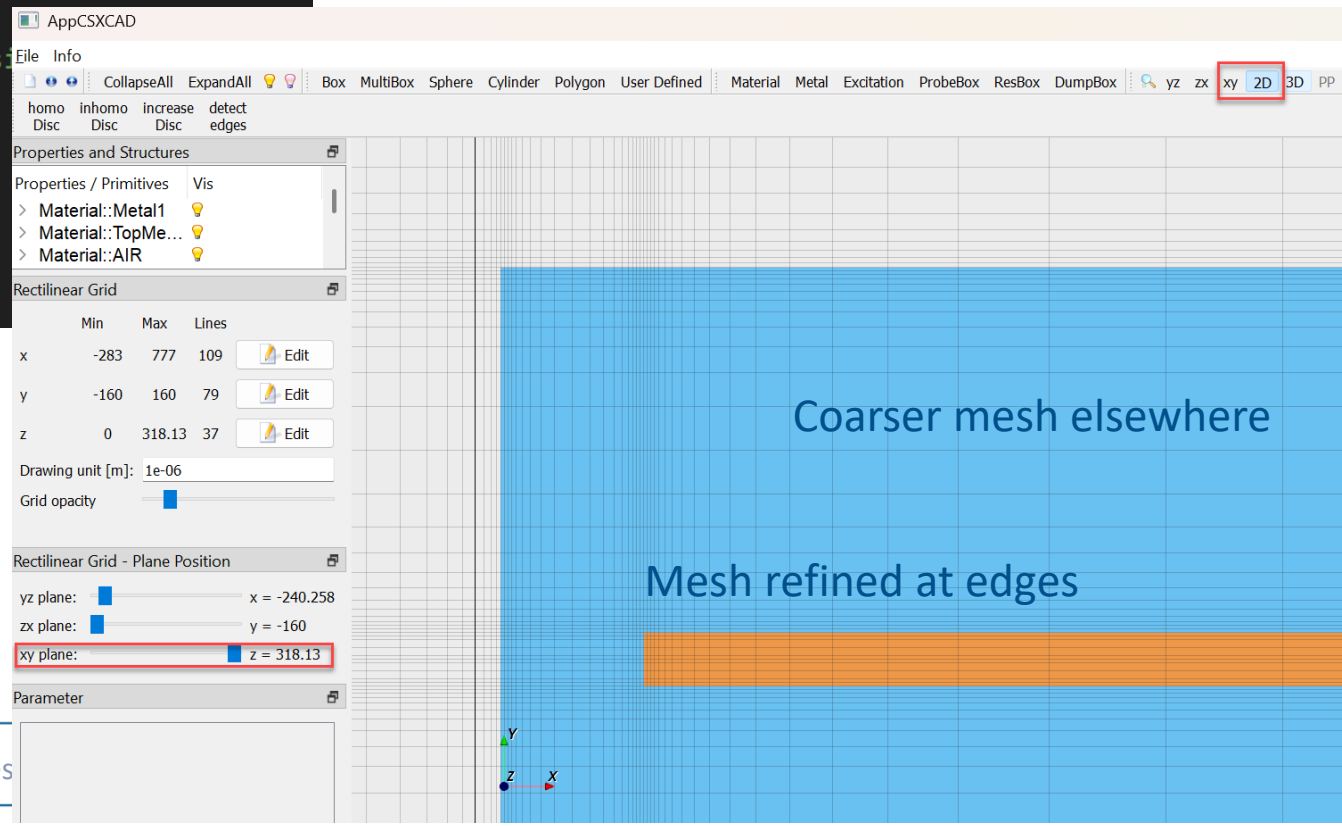



```python
# ====================== input files and path settings =======================

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)
```

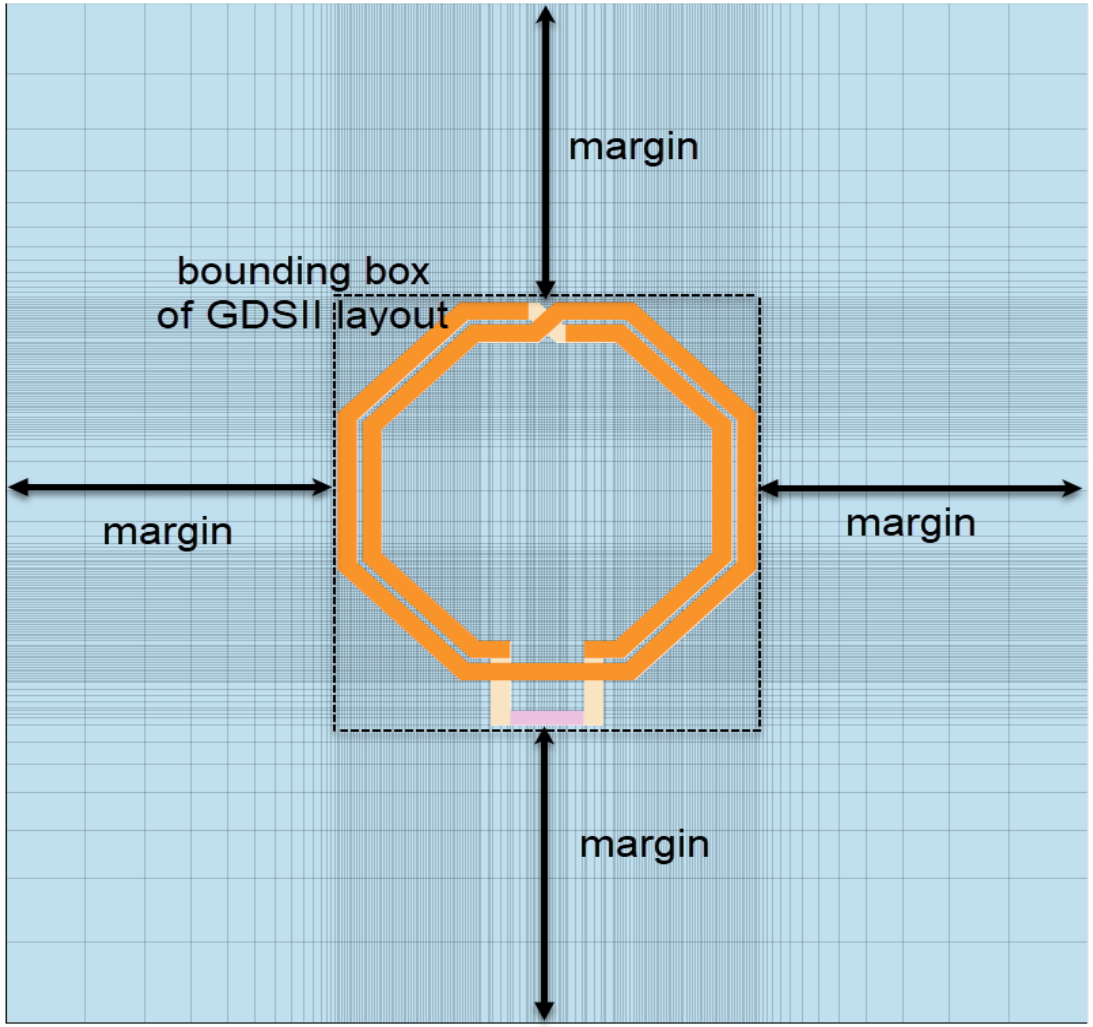
Simulation control

- Mesh is built automatrically based on geometry
- Only one parameter to set required minimum mesh size (required detail resolution)

```
unit    = 1e-6  # geometry is in microns
margin = 50    # distance in microns from GDSII geometry boundary to si

fstart =  0e9
fstop  = 110e9
numfreq = 401

refined_cellsize = 1  # mesh cell size in conductor region
```



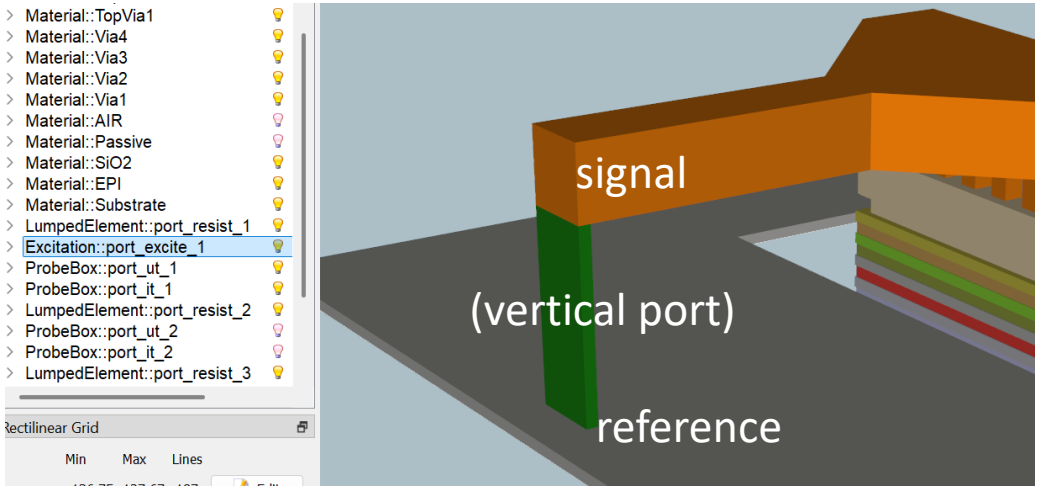
Margins and simulation boundary

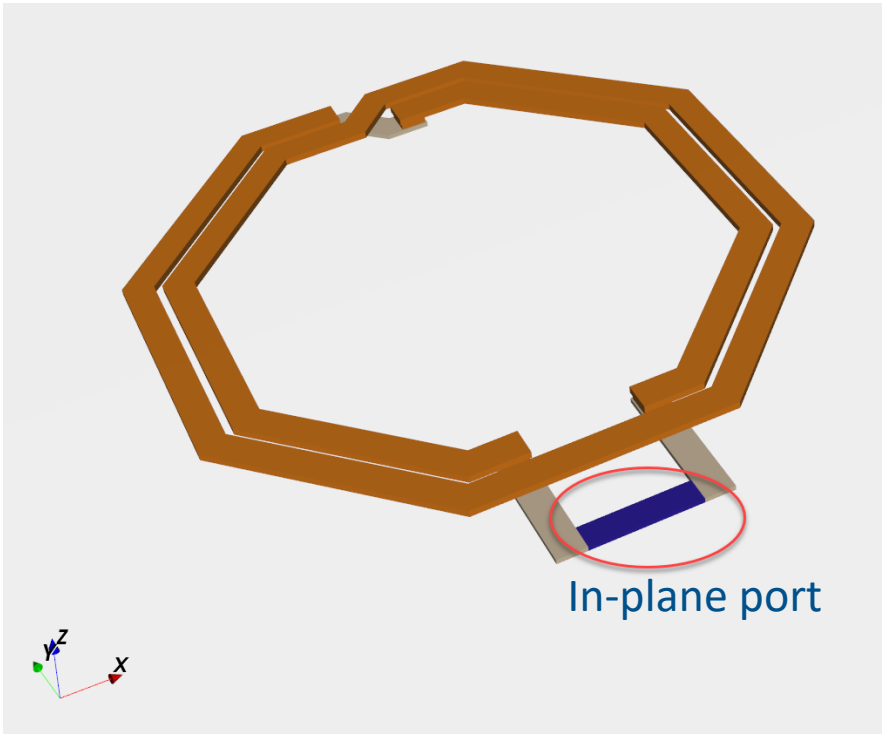



```
# ======================== simulation settings ================================

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)
```

Ports: in-plane or vertical between layers

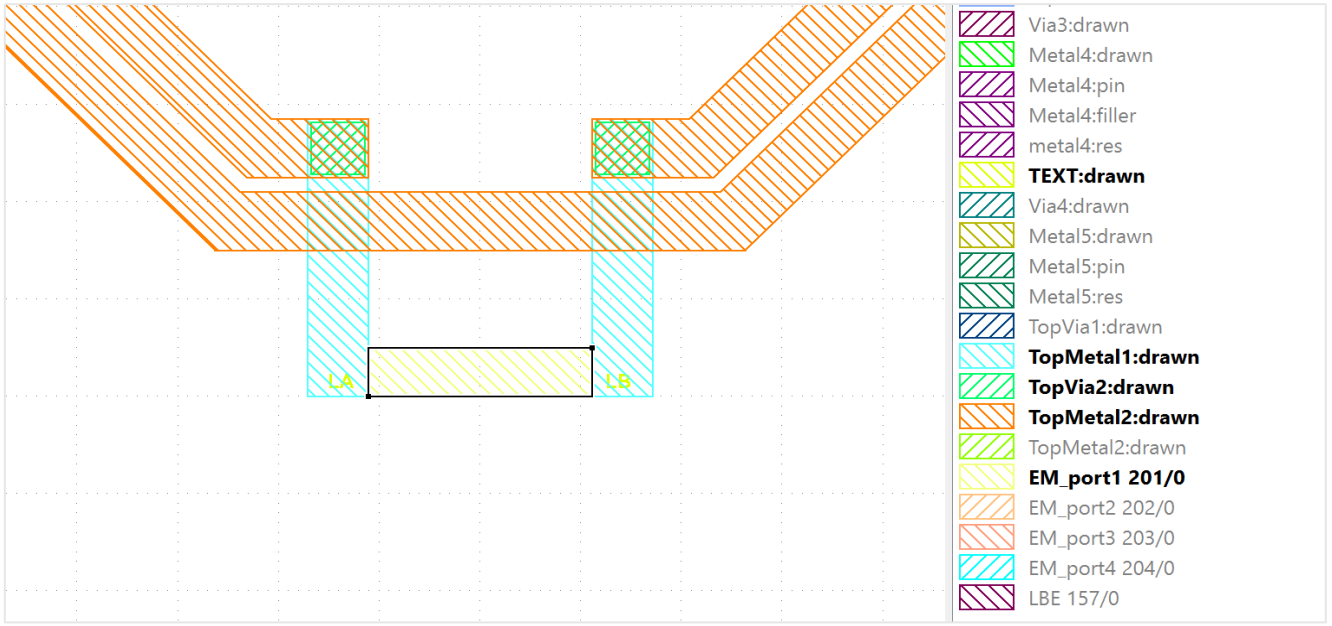
- In-plane ports between metals on same layer
- Vertical ports between metals of different layers





Port configuration in new workflow

- Port shape defined in GDSII on special layer (recommended 201 and above)
- Separate layer per port
- PDK pins from GDSII not used because they have wrong shape and position



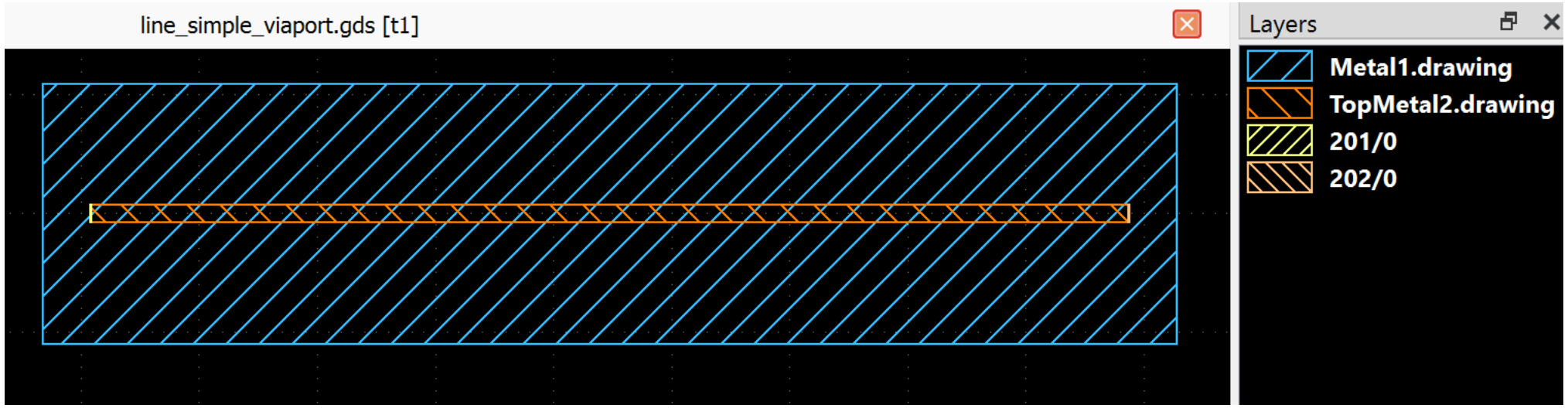

```
# ports from GDSII Data, polygon geometry from specified special layer
# note that for multiport simulation, excitations are switched on/off in simulation_setup.createSimulation below
simulation_ports = simulation_setup.all_simulation_ports()
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1,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

Overview

1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Typical simulation run
5. Automatic Meshing Algorithm
6. Simulated vs. measured
7. Summary

Simple model: Transmission line

- Geometry: TopMetal2 transmission line with Metal1 ground plane
- Metal1 ground plane must be drawn to include it in model
- Input port 1 drawn as small box on layer 201
- Output port 2 drawn as small box on layer 202



Code from run_line_viaport.py

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

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

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

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

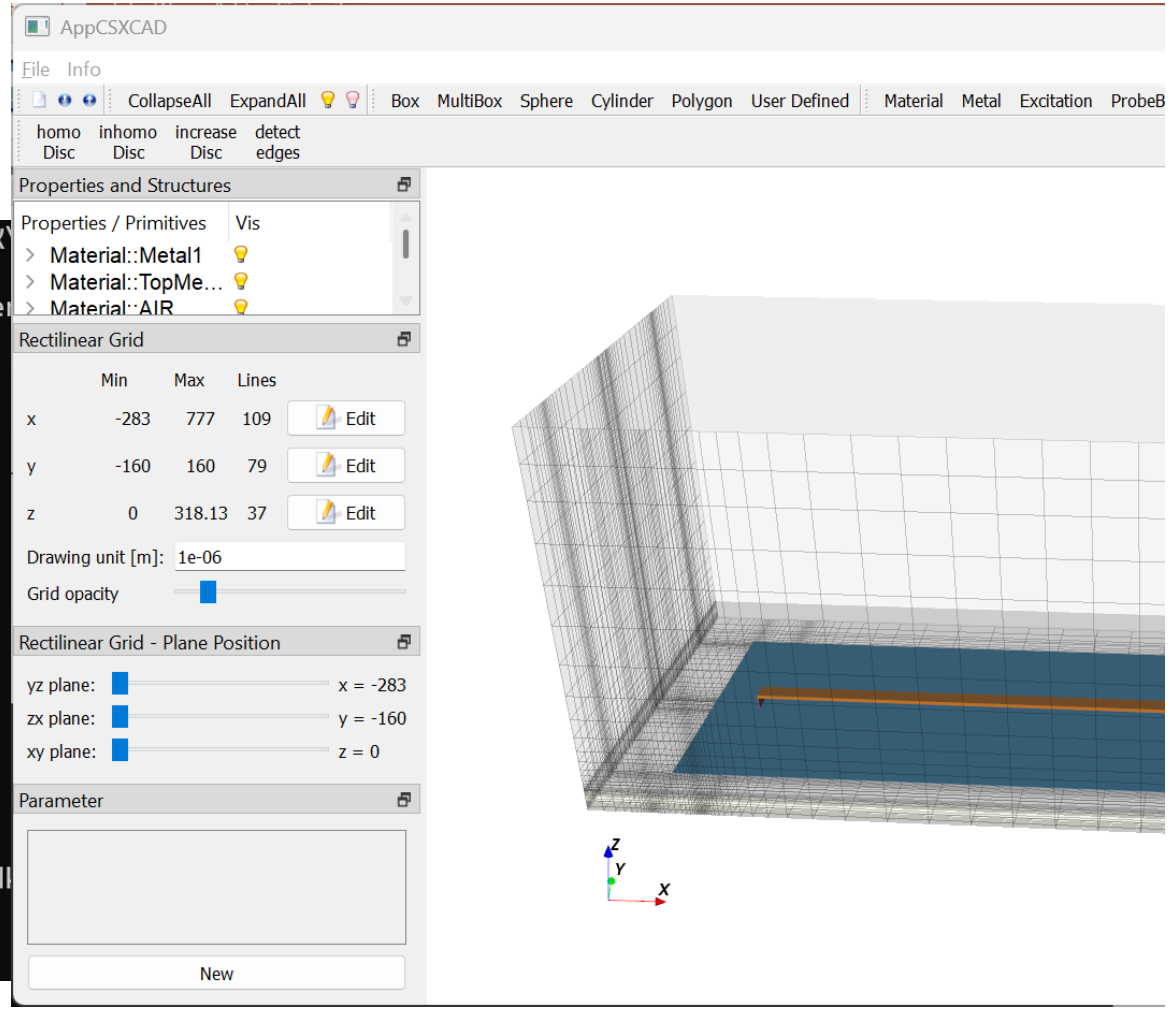
```
unit    = 1e-6  # geometry is in microns
margin = 50    # distance in microns from GDSII geometry boundary to simulation boundary

fstart =  0e9
fstop  = 110e9
numfreq = 401

refined_cellsize = 1  # 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 = -40          # end criteria for residual energy (dB)
```

```
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
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'))
```

Run model

- Change to model directory
- From command line: `python3 run_line_viaport.py`

```
D:\perforce\volker_OMEN15_7389\volker_OMEN15_7389\OpenEMS_Python_OPDK\
t.py
Simulation data directory:  d:\perforce\volker_omen15_7389\volker_ome
_ihp\output\run_line_viaport_data
Reading XML stackup  file: SG13G2_nosub.xml
Reading GDSII input file: line_simple_viaport.gds


------------------------
Mesh cells by axis (total 327 kcells):
 x = 109
 y = 79
 z = 38
Smallest cell size:
 dx = 1.0000
 dy = 1.0000
 dz = 0.4000
------------------------

Starting AppCSXCAD 3D viewer with file:
 d:\perforce\volker_omen15_7389\volker_omen15_7389\openems_python_opdk
t_data\sub-1\run_line_viaport.xml
Close AppCSXCAD to continue or press <Ctrl>-C to abort
```



FDTD Simulation (finished)

- Simulation of this port excitation is finished when residual energy is below limit defined in code

```
C:\Windows\System32\cmd.e
Create FDTD operator (compressed SSE + multi-threading)
FDTD simulation size: 109x79x37 --> 318607 FDTD cells
FDTD timestep is: 1.70744e-15 s; Nyquist rate: 2662 timesteps @1.10006e+11 Hz
openEMS::SetupFDTD: Warning, the timestep seems to be very small --> long simulati
Excitation signal length is: 30506 timesteps (5.20873e-11s)
Max. number of timesteps: 1000000000 ( --> 32780.4 * Excitation signal length)
Create FDTD engine (compressed SSE + multi-threading)
Running FDTD engine... this may take a while... grab a cup of coffee?!?
[@        4s] Timestep:          665 || Speed:   51.0 MC/s (6.243e-03 s/TS) || Ene
[@       10s] Timestep:         1995 || Speed:   62.3 MC/s (5.111e-03 s/TS) || Ene
[@       15s] Timestep:         3325 || Speed:   92.0 MC/s (3.461e-03 s/TS) || Ene
[@       20s] Timestep:         5320 || Speed:  120.1 MC/s (2.653e-03 s/TS) || Ene
[@       25s] Timestep:         7315 || Speed:  139.5 MC/s (2.285e-03 s/TS) || Ene
[@       30s] Timestep:         9975 || Speed:  161.0 MC/s (1.979e-03 s/TS) || Ene
[@       35s] Timestep:        12635 || Speed:  177.5 MC/s (1.795e-03 s/TS) || Ene
[@       39s] Timestep:        15253 || Speed:  187.7 MC/s (1.697e-03 s/TS) || Ene
[@       44s] Timestep:        17955 || Speed:  190.9 MC/s (1.669e-03 s/TS) || Energy: ~2.23e-18 (- 0.00dB)
[@       48s] Timestep:        20615 || Speed:  202.4 MC/s (1.574e-03 s/TS) || Energy: ~4.62e-19 (- 6.84dB)
[@       52s] Timestep:        23275 || Speed:  204.6 MC/s (1.557e-03 s/TS) || Energy: ~2.69e-19 (- 9.19dB)
[@       57s] Timestep:        25935 || Speed:  169.4 MC/s (1.880e-03 s/TS) || Energy: ~4.15e-21 (-27.30dB)
Multithreaded Engine: Best performance found using 11 threads.
[@     1m01s] Timestep:        28595 || Speed:  198.4 MC/s (1.606e-03 s/TS) || Energy: ~1.04e-21 (-33.33dB)
[@     1m06s] Timestep:        31255 || Speed:  199.8 MC/s (1.595e-03 s/TS) || Energy: ~3.70e-22 (-37.81dB)
[@     1m10s] Timestep:        33915 || Speed:  208.5 MC/s (1.528e-03 s/TS) || Energy: ~2.73e-22 (-39.12dB)
[@     1m14s] Timestep:        37240 || Speed:  232.0 MC/s (1.373e-03 s/TS) || Energy: ~1.94e-22 (-40.61dB)
Time for 37240 iterations with 318607.00 cells : 74.87 sec
Speed: 158.46 MCells/s
FDTD simulation completed successfully for excitation  [1]
Creating  S-parameter file
```

```
fstart =  0e9
fstop  = 110e9
numfreq = 401

refined_cellsize = 1  # 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 = -40          # end criteria for residual energy (dB)
```

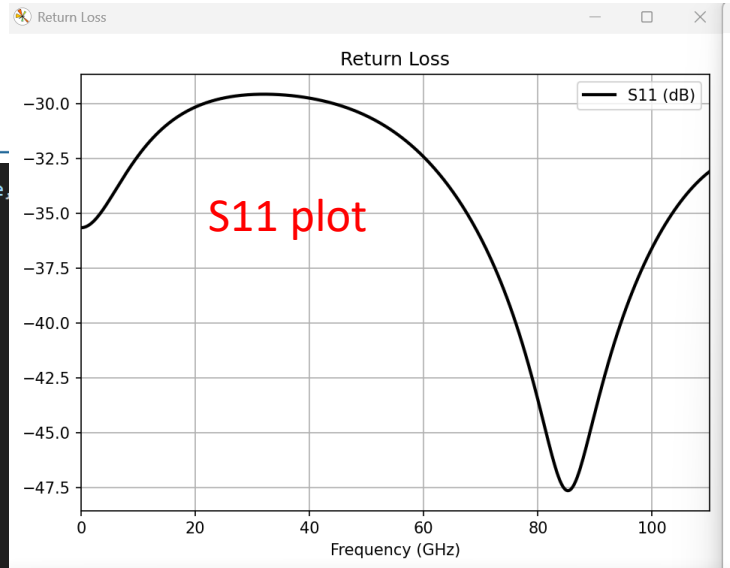
Data plot, create data file

```
sub1_data_path = simulation_setup.runSimulation (excite_ports, FDTD, sim_path, model_basename,


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

if preview_only==False:

    # 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 np.angle(value, deg=True)


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

    # get results, CSX port definition is read from simulation ports object
    s11 = utilities.calculate_Sij (1, 1, f, sim_path, simulation_ports)
    s21 = utilities.calculate_Sij (2, 1, f, sim_path, simulation_ports)

    # 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)

    fig, axis = plt.subplots(num='Return Loss', tight_layout=True)
    axis.plot(f/1e9, dB(s11), 'k-',  linewidth=2, label='S11 (dB)')
    axis.grid()
    axis.set_xmargin(0)
    axis.set_xlabel('Frequency (GHz)')
    axis.set_title('Return Loss')
    axis.legend()
```

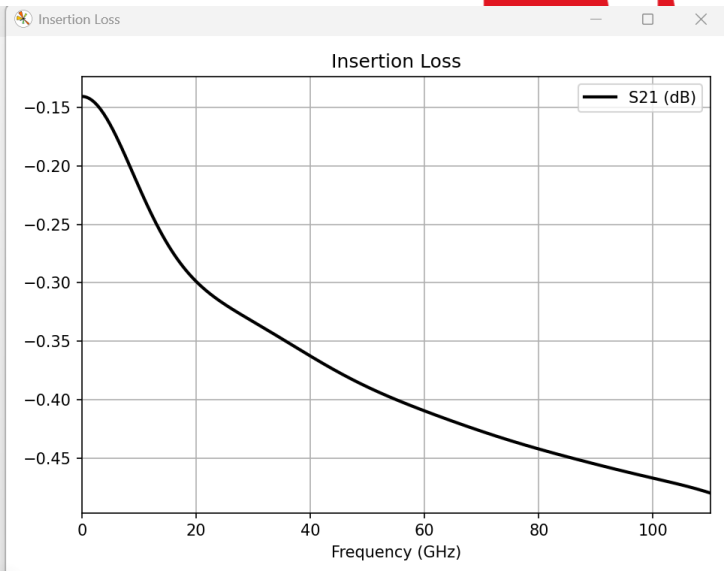
fake symmetry

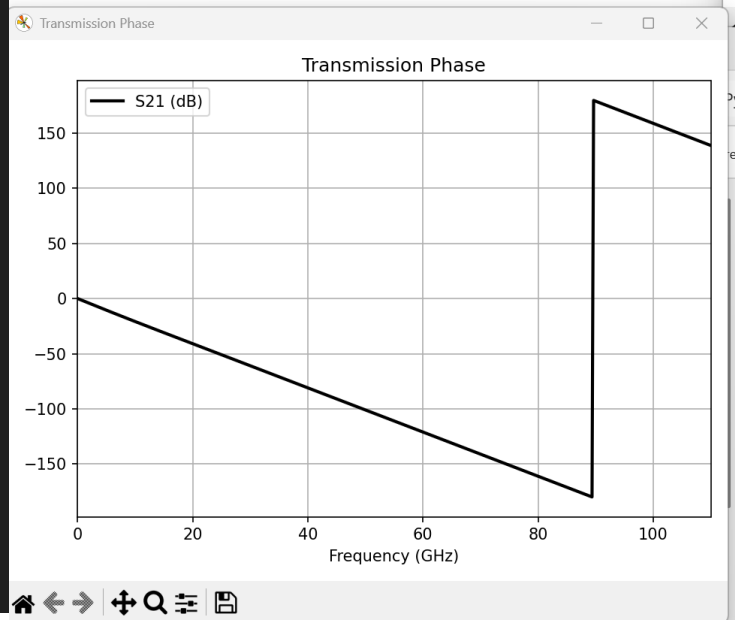
write S2P file

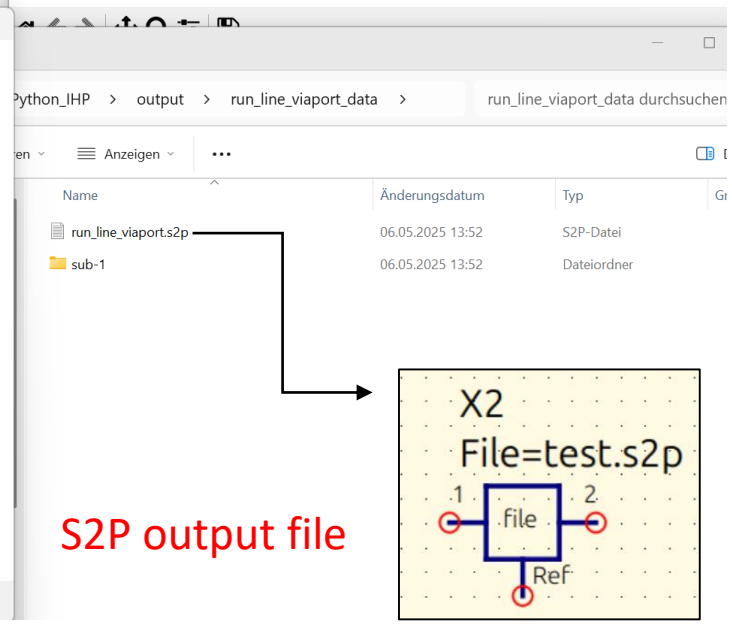
S11 plot



S11 plot







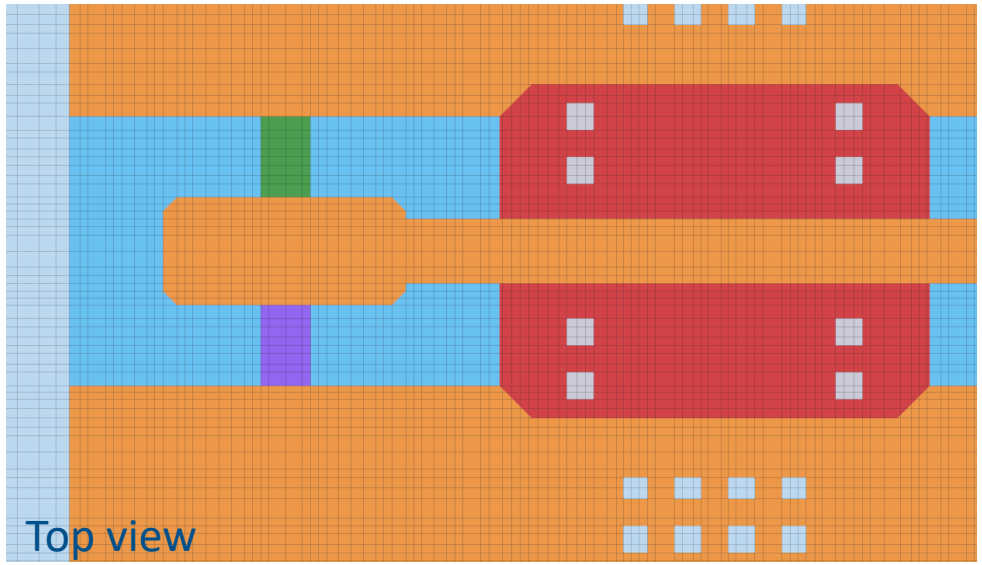
S2P output file

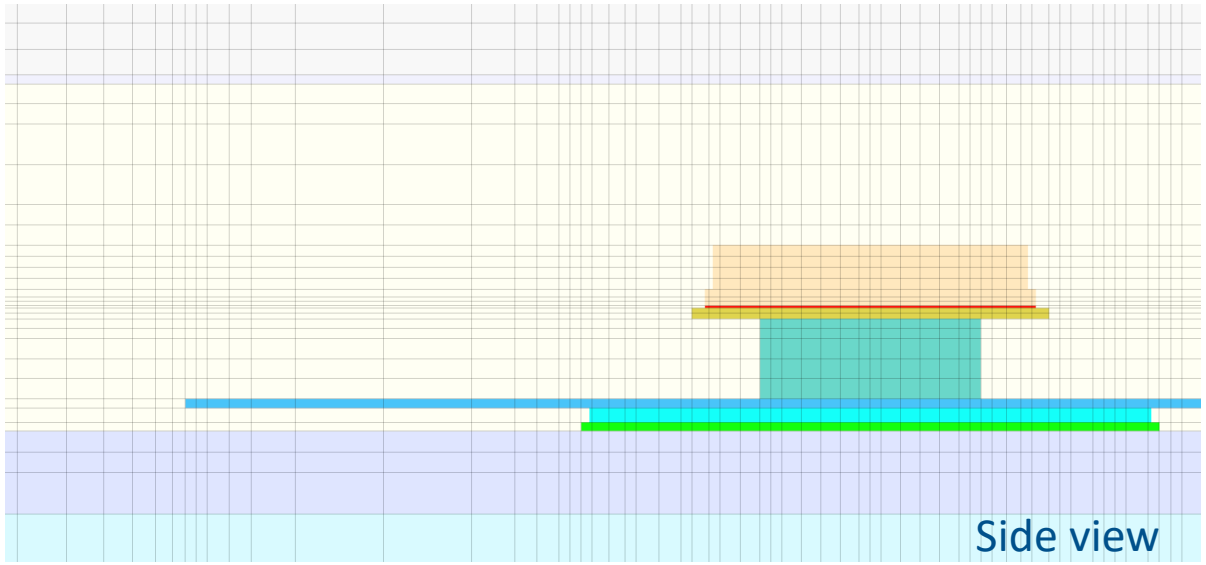
Overview

1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Typical simulation run
5. Automatic Meshing Algorithm
6. Simulated vs. measured
7. Summary

Meshing goals and requirements

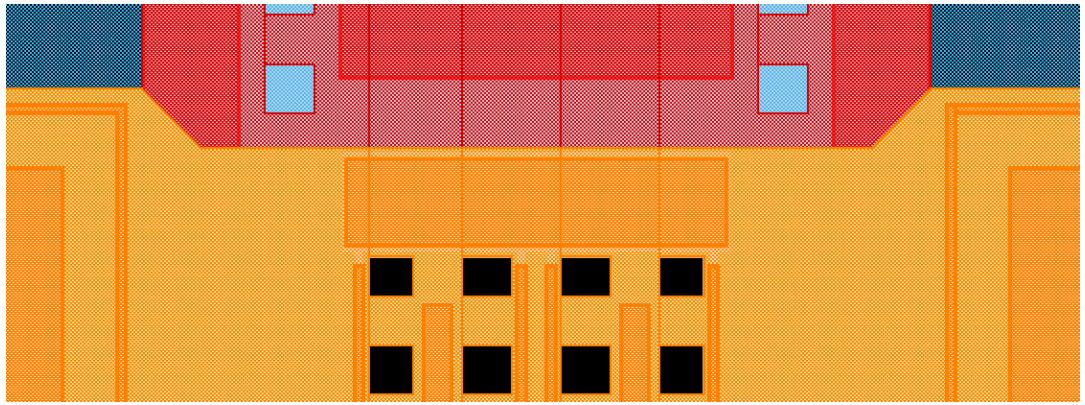
- Different from FEM method, FDTD uses fixed mesh. No adaptive mesh refinement during simulation.
- Skin effects pushes current to conductor edges, refine mesh there
- Larger mesh cells inside, not much variation of current flow there
- Upper limit for mesh cell size given by wavelength, at least 10 cells per wavelength required
- Smallest „target“ mesh size defined by user

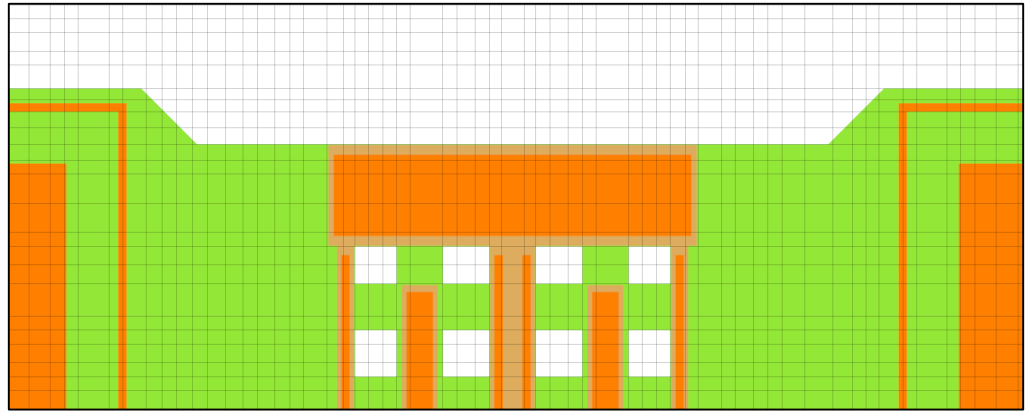




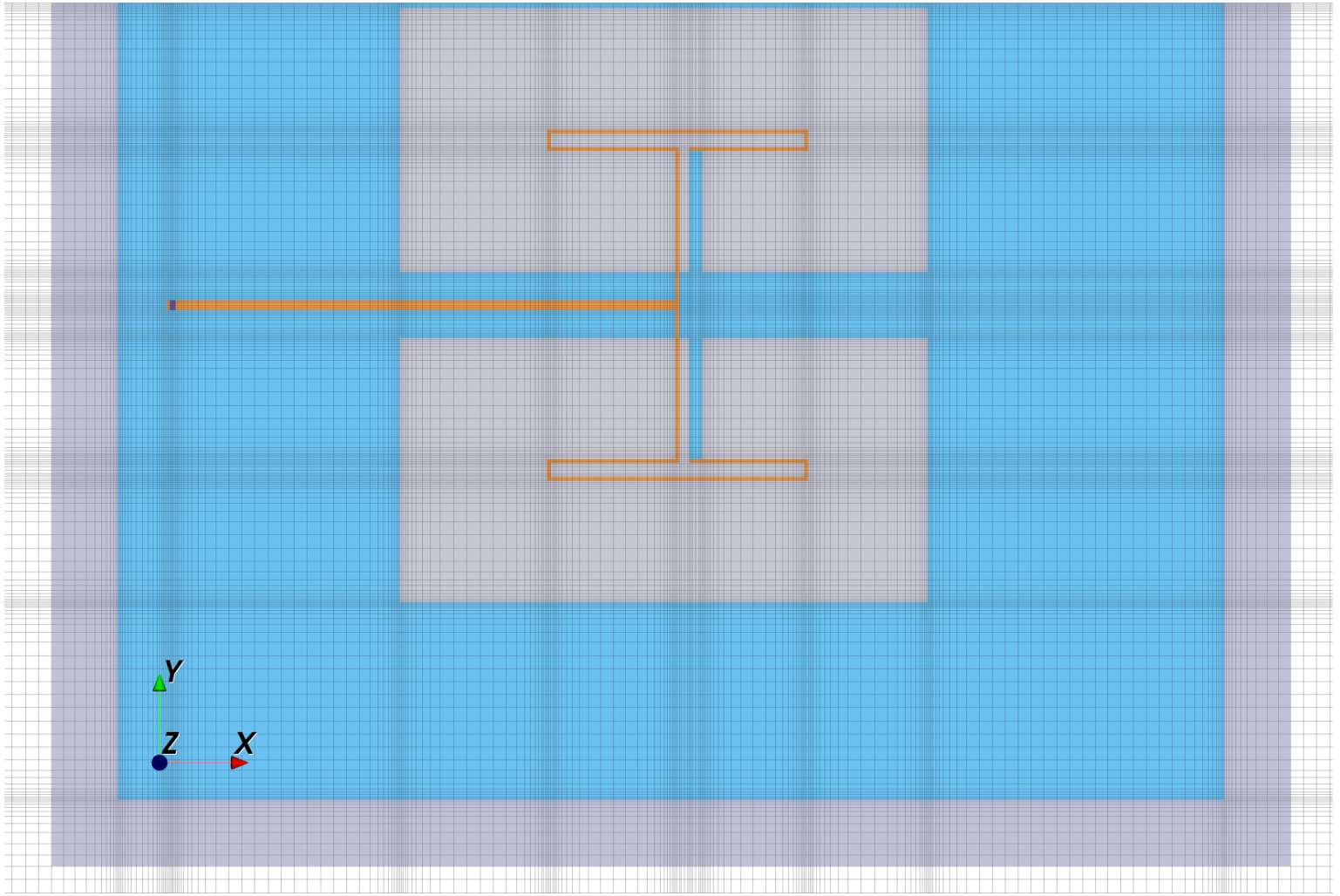
Create xy mesh from polygons

- Step 1: Create lines at all polygon edges, with different „weight“ for different polygon types: port edge = 20, polygon edge = 10, via edge =5
- Step 2: Place extra mesh lines along diagonal edges, these fill lines have weight = 1
- Step 3: Remove mesh lines that are too close, replace by one combined mesh line, position based on weight of lines that we merge together here
- Step 4: Add intermediate mesh lines in large mesh cells
- Step 5: Check for possible remaining gaps, add lines required for smooth mesh





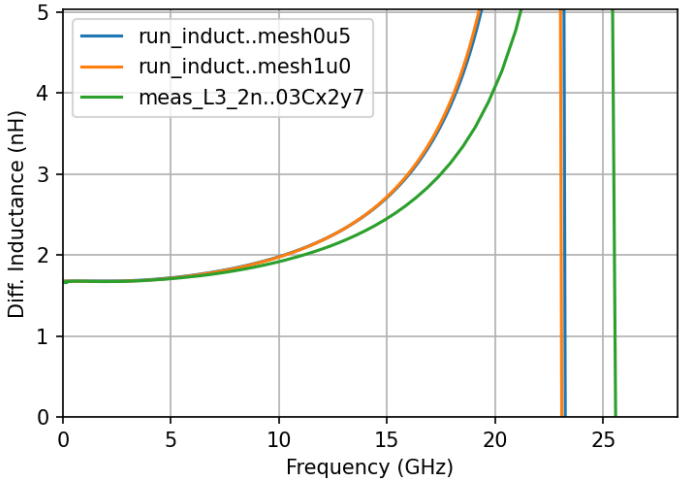
Antenna mesh example

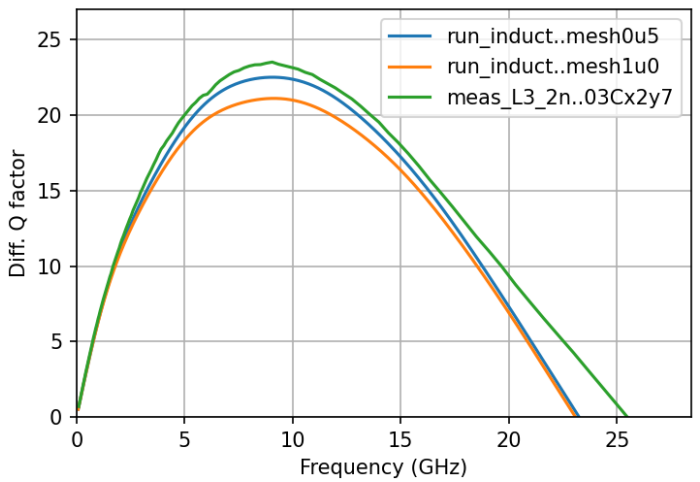


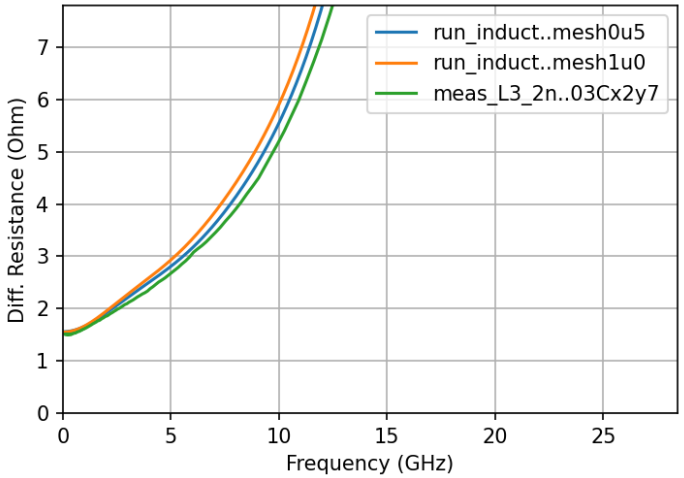
Overview

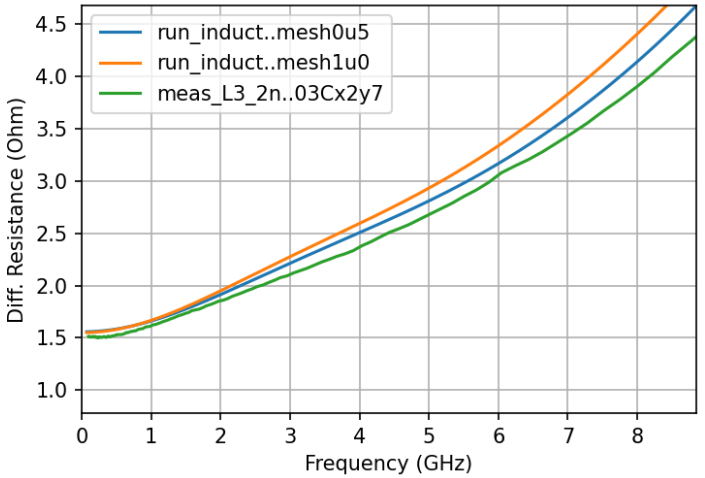
1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Typical simulation run
5. Automatic Meshing Algorithm
6. Simulated vs. measured
7. Summary

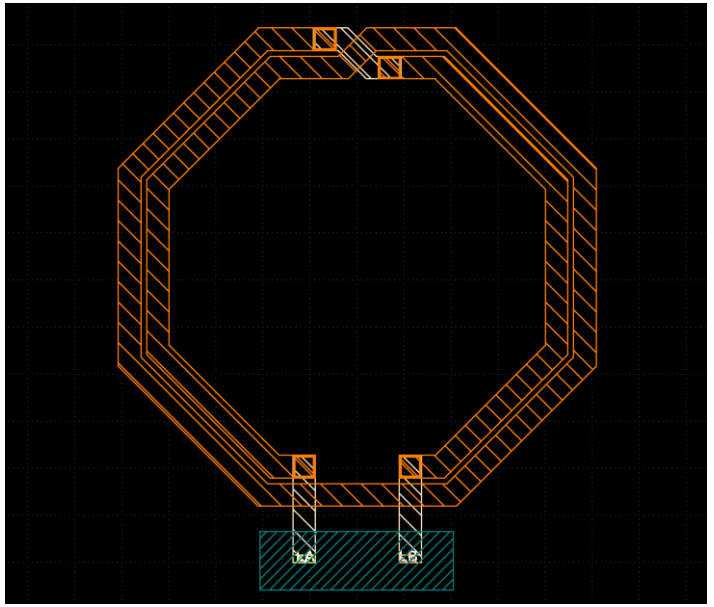
Simulation vs. Measured: Inductor 2nH @ 10 GHz

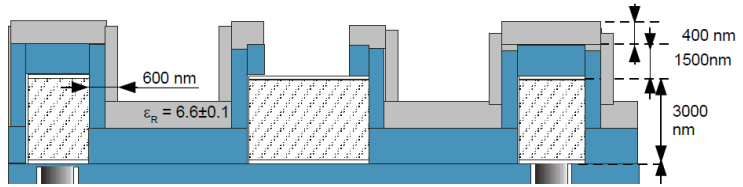












Limitation: EM model uses planar stackup
Conformal passivation has some air between traces
Commercial EM with planar stackup: SRF = 23.8 GHz

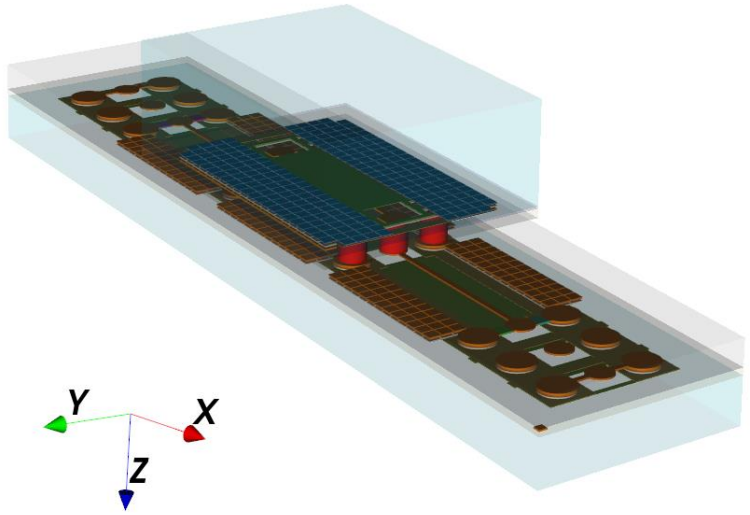
Overview

1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Typical simulation run
5. Automatic Meshing Algorithm
6. Simulated vs. measured
7. Summary

Summary and Outlook

- openEMS is time domain EM simulation, results are the usual *.snp Touchstone files
- New workflow reads geometries from GDSII, models are much easier to re-use
- Simulation port geometry defined in GDSII file on extra layers (recommended 201 and above)
- Port parameters are set in Python model file, pay attention to port direction and polarity
- Simulation mesh requires only one user input, to set minimum mesh cell size

- Work in progress: add support for multi-chip module simulation
- Work in progress: create similar workflow for FEM instead of FDTD



References

- openEMS by Thorsten Liebig et al:
  https://www.openems.de/
- New IHP workflow for GDSII files
  https://github.com/VolkerMuehlhaus/openems_ihp_sg13g2
- Lumped model extraction from S2P files:
  https://github.com/VolkerMuehlhaus/lumpedmodel

- Detailed manual for new workflow:
  https://github.com/VolkerMuehlhaus/openems_ihp_sg13g2/blob/main/doc/Using_OpenEMS_Python_with_IHP_SG13G2_v2.pdf

Thank you for your attention!

IHP GmbH – Leibniz Institute for High Performance Microelectronics
Im Technologiepark 25
15236 Frankfurt (Oder)

E-Mail: volker@muehlhaus.com