50 GHz Medium Power Amplifier: Exploring Qucs-s & OpenEMS

Phillip Ferreira Baade–Pedersen (Design Engineer)
Dr. Ing. Christian Wittke (Scientist)
Frankfurt (Oder), IHP - 21/05/2025

Projects: BMBF -> FMD-QNC (16ME0831)

Leibniz Institute for high performance microelectronics

Agenda For today

09:00 - 09:15 | Recap | Questions?
Review of Key Takeaways from Yesterday

09:15 – 09:30| Module Overview
Overview and Introduction to Today's Module

09:30 - 10:45 | Expert Talk
Dr.-Ing Volker Muehlhaus: *Python Interfacing with OpenEMS using IHP Stackup*

10:45 - 12:00 | Introduction | Hands On
Introduction To QUCS-S / Starting the MPA design

12:00 - 13:00 | Lunch Brake | Catching Up

13:00 - 15:00 | Design of 50 GHz MPA
Build a simple schematic and set up the RF design flow, including S-parameter and nonlinear analysis. (Starting EM simulations)

15:00 – 15:30 | Coffee Break | Caching Up

15:30 - 17:00 | EM Simulations
Perform EM simulation of schematic components and analyze their impact on circuit behavior

Leibniz Institute for high performance microelectronics

Key Takeaways from Yesterday

- Insights on models within the IHP Open PDK
- Building the testbench for two stage OTA
- Building the testbench for the BGR
- Exploring mismatch simulations
- Beginning the layout of two stage OTA

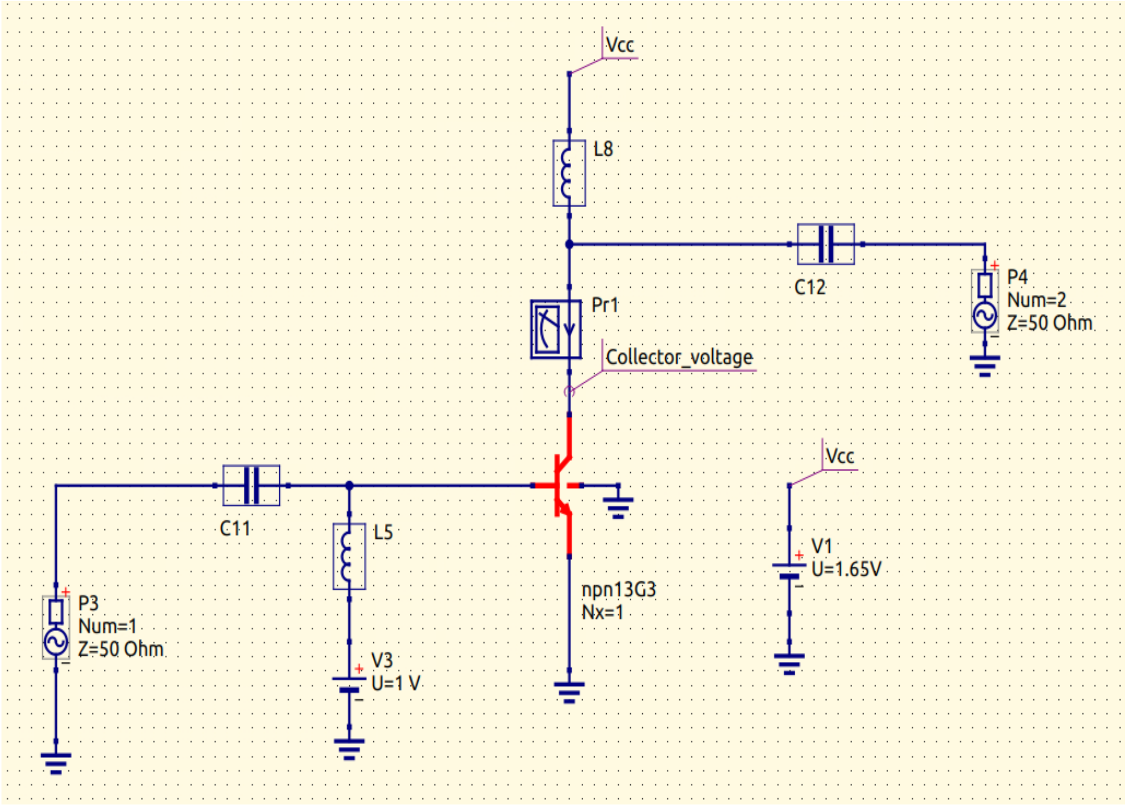
Questions?

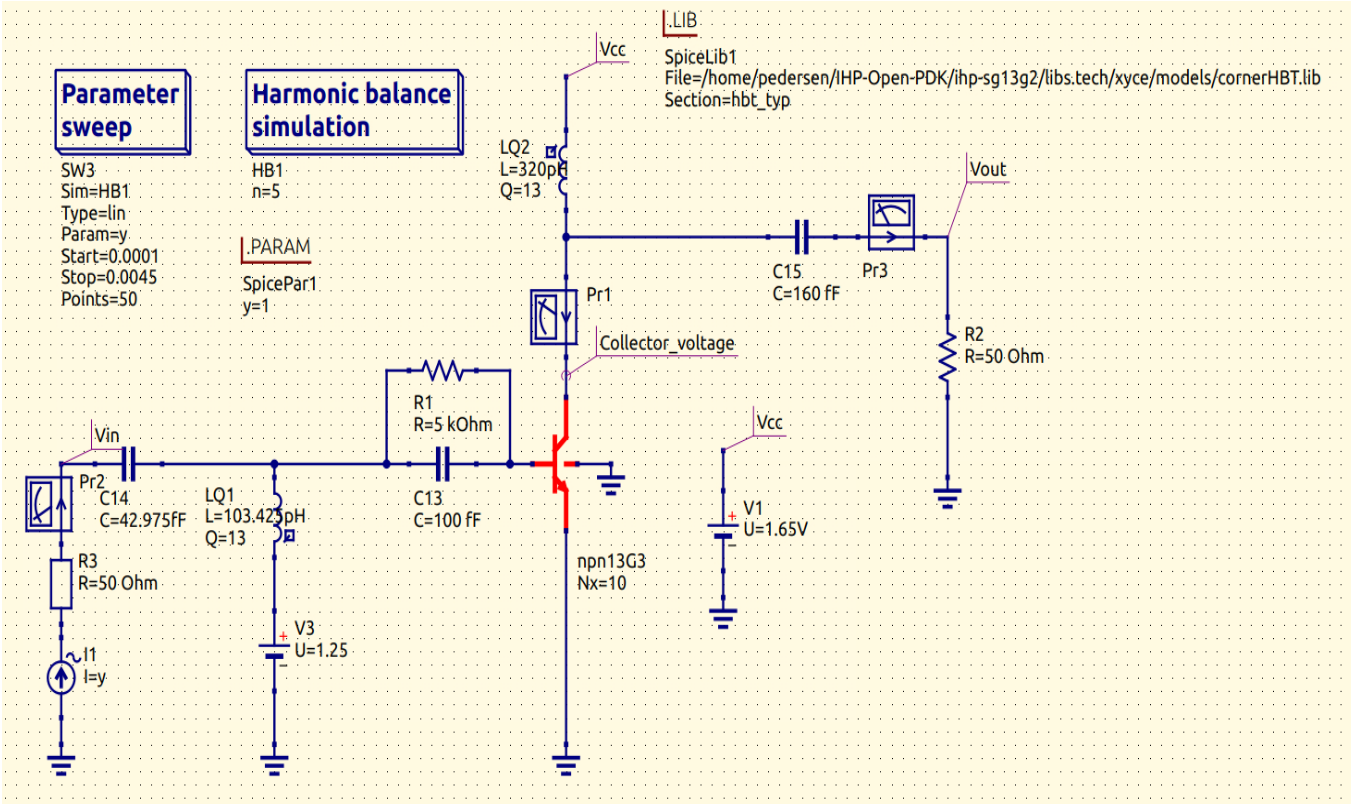
- Was something from yesterday unclear?
- Are we going to fast?
- Do you need more time for the small exercise?

Overview Of Todays Plan

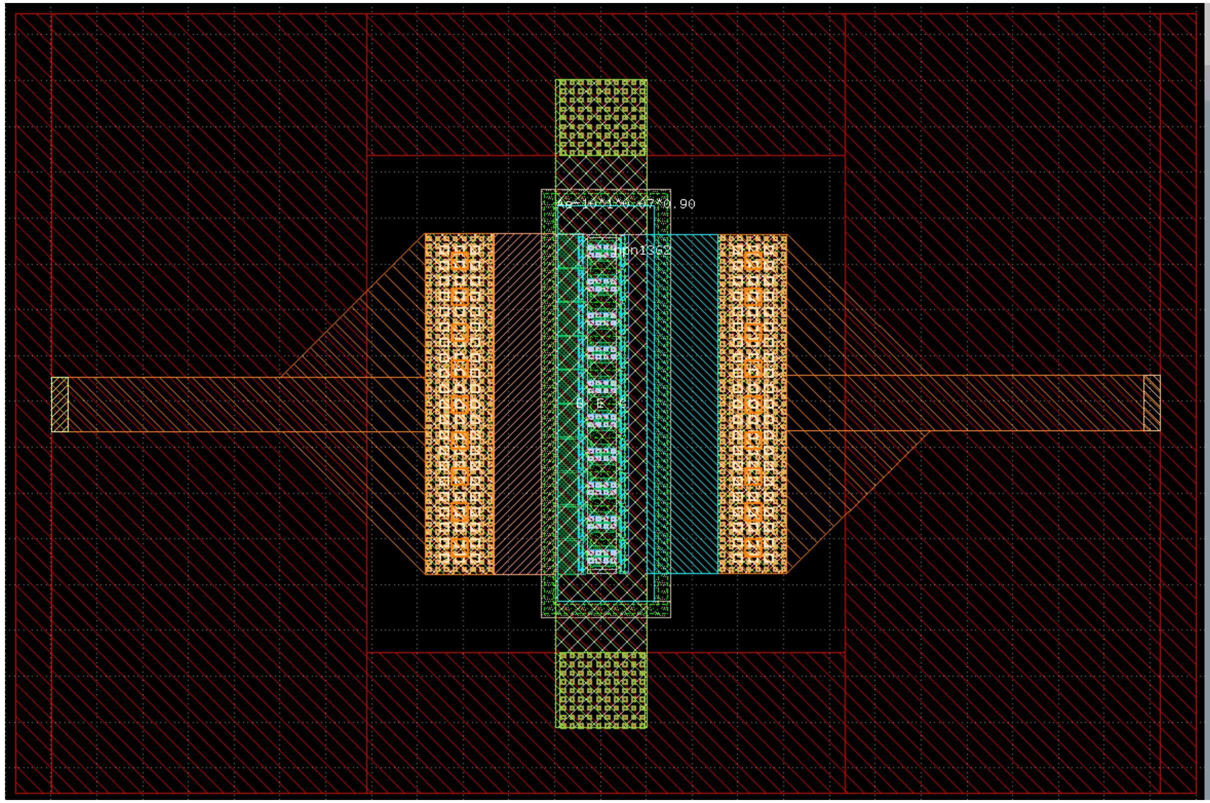
- **Exploring QUCS-S:** Small introduction to Qucs-s
- **MPA Schematic:** Biasing and Initial matching
- **Compression Point:** Non-Linear analysis using xyce to determine compression point
- **OpenEMS:** EM Simulation using OpenEMS
- **Misc:** Continue OTA layout or perform EM simulations for the remaining MPA components.

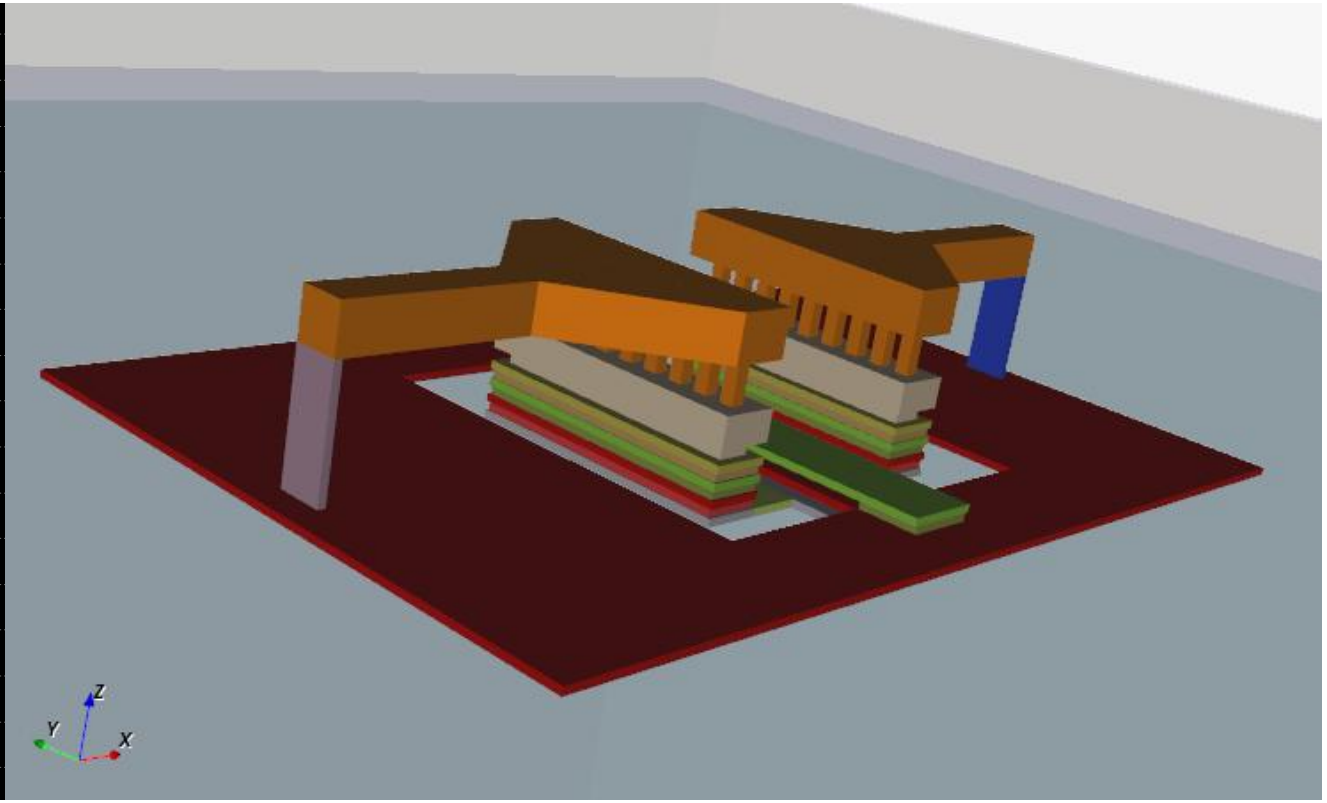
Single Stage 50 GHz MPA



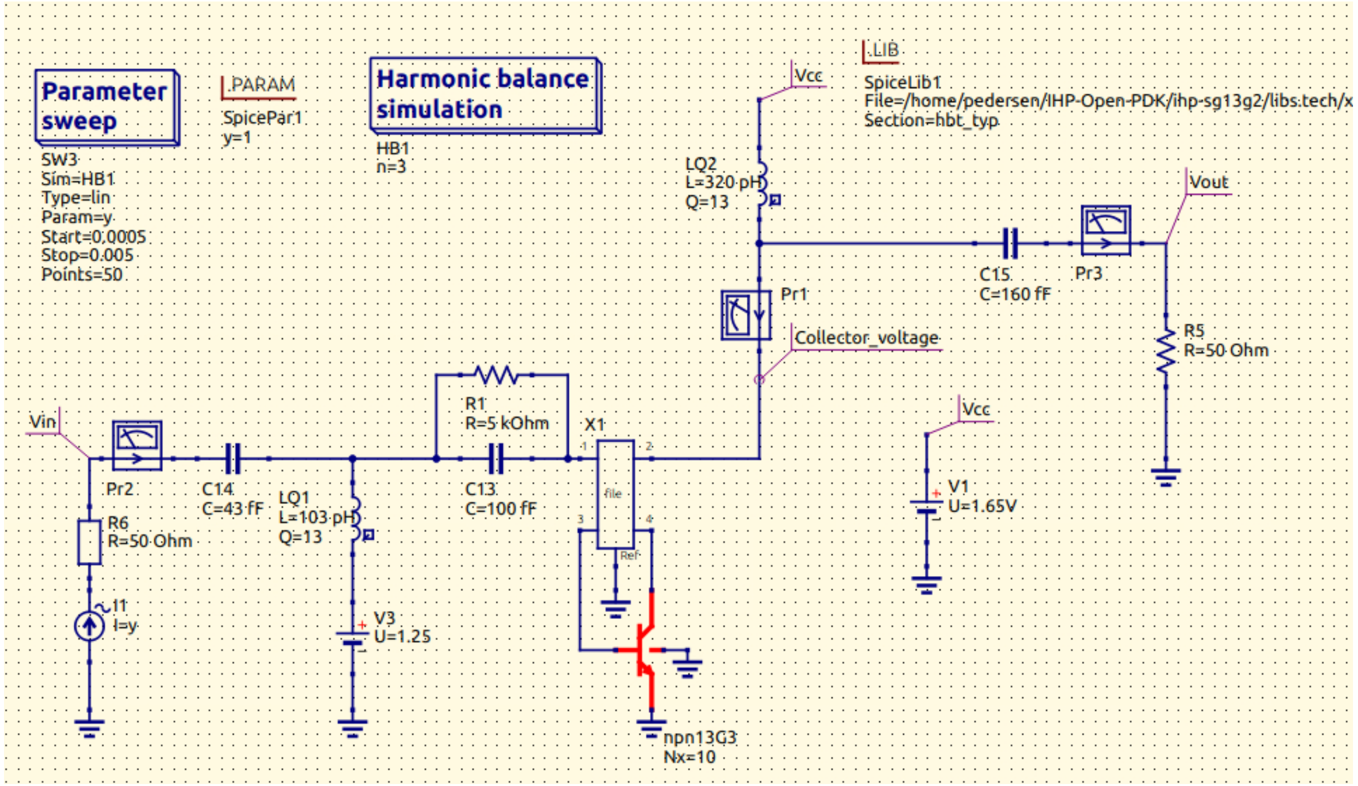


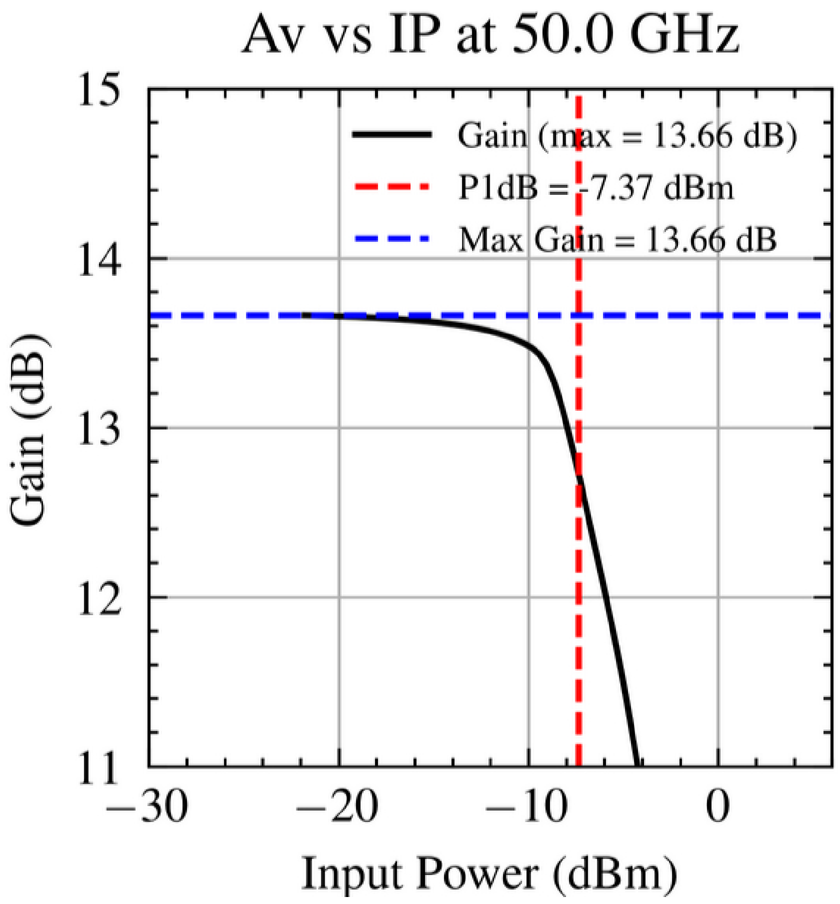
OpenEMS





Post Processing





Small Disclaimer

RF Workflow in OS is challenging !

- Harmonic Balancing in Xyce is challenging… Why? (sweeps, equations etc.)
- This workflow propose a procedure to extract results and metrices in a semi automated way
- RF workflow is based on repetition so be patient ☺

Good News?

- The OS echo system is pushing for a lot of development in this area
- EM simulation with python interfacing is convenient and performs well!
- For ***HB*** simulations Ngspice developers are currently implementing this

Expert Talk

Dr.-Ing Volker Muehlhaus

Python Interfacing with OpenEMS using IHP Stackup

Leibniz Institute
for high
performance
microelectronics

Leibniz Institute for high performance microelectronics

EM Simulation with openEMS for SG13G2 (new workflow)

Volker Muehlhaus

05 May 2025

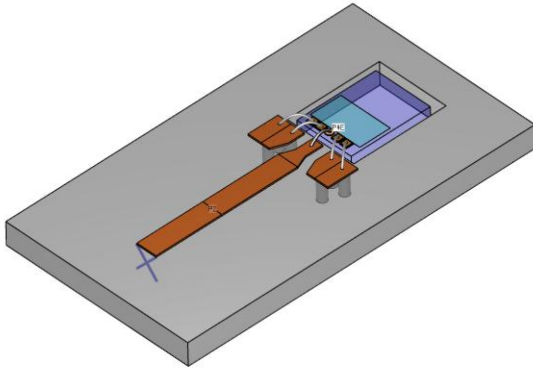
Overview

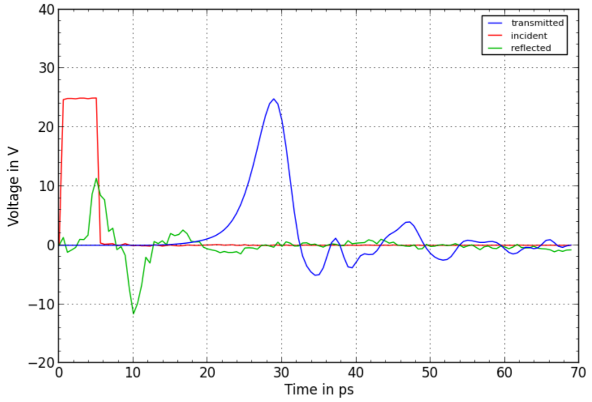
1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Examples transmission line and PA core
5. Simulation mesh
6. Using S-Parameter results, optional lumped model extraction
7. Summary

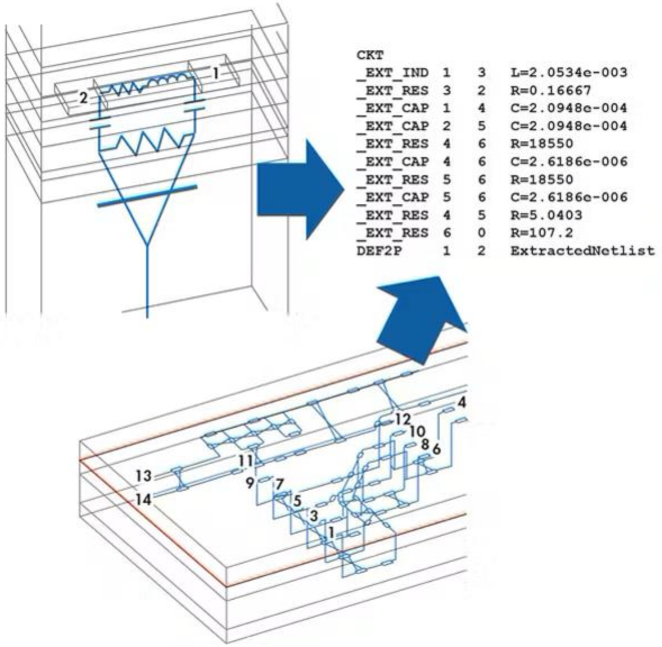
Why use EM simulation?

- Interconnect layout adds series R,L and shunt C
- No electrical model of layout/interconnect in the PDK
- No parasitic extraction available in openPDK flow
- EM simulation can capture ALL layout effects,
  valid up to highest frequencies,
  but requires more complex analysis
- For IHP Open PDK, we can use EM solver openEMS

Parasitic Extraction
not available in openPDK







https://www.mwrf.com/technologies/embedded/software/article/21846943/em-simulation-technologies-support-rfic-development

EM simulation use cases

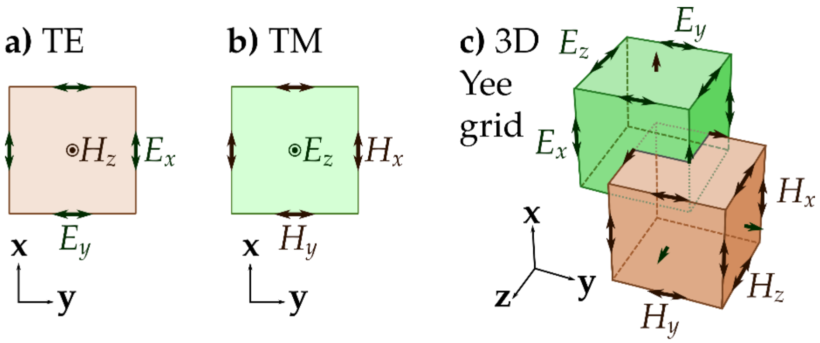
- Interconnect parasitics
- Layout components:
  - Inductors, Baluns
  - MIM
  - Transmission lines

Overview

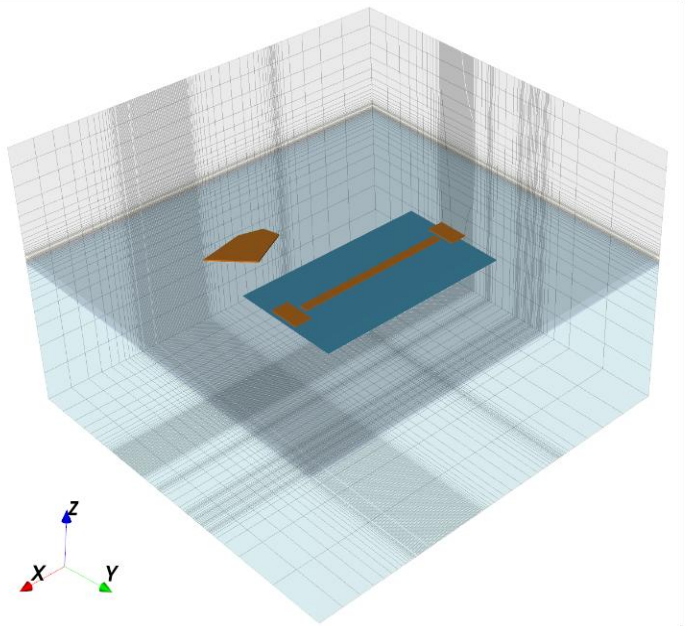
1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Examples transmission line and PA core
5. Simulation mesh
6. Using S-Parameter results, optional lumped model extraction
7. Summary

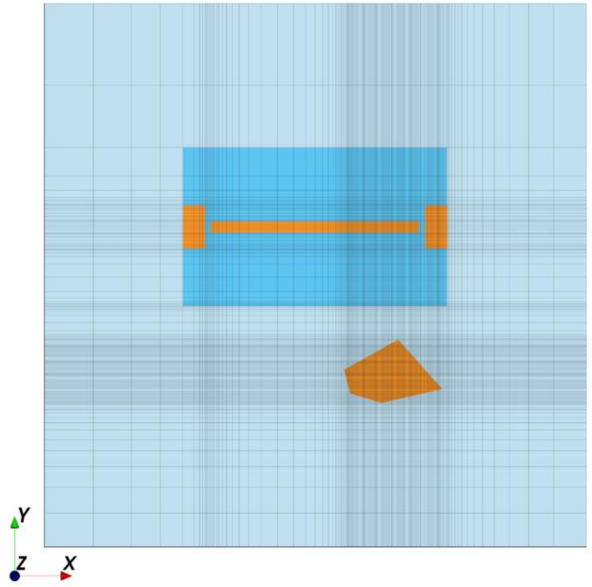
FDTD Method used in openEMS: Mesh

- 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…
- Simulation boundary can be perfect electric conductor, perfect magnetic conductor or absorbing



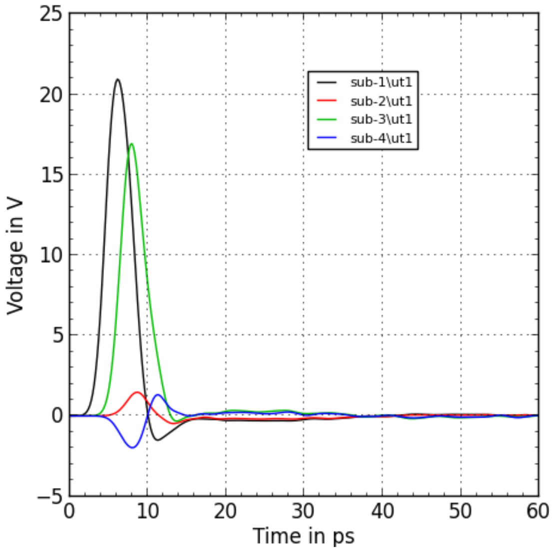
Von FDominec - Eigenes Werk, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=37637085

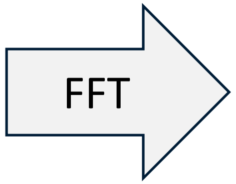




FDTD Method used in openEMS: Time Domain

- Port excitation with gaussian pulse creates wideband continuous spectrum
- FDTD calculates E and H in time domain, step by step, until energy in model is (near) zero
- Wideband S-Parameter obtained by Fourier transform of time signals at port(s)
- We get one column of S-matrix (wideband) per port excitation
- Repeat for all ports to get full [S] matrix





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

Overview

1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Examples transmission line and PA core
5. Simulation mesh
6. Using S-Parameter results, optional lumped model extraction
7. Summary

Basis of this work: openEMS

https://www.openems.de

openEMS is a free and open electromagnetic field solver using the FDTD method.

- Documentation
- Legacy Wiki
- Gihub Discussions
- Imprint

Welcome! openEMS is a free and open electromagnetic field solver using the FDTD method. Matlab or Octave and Python are used as an easy and flexible scripting interface.

Install

Latest Windows Build

Linux Build instructions

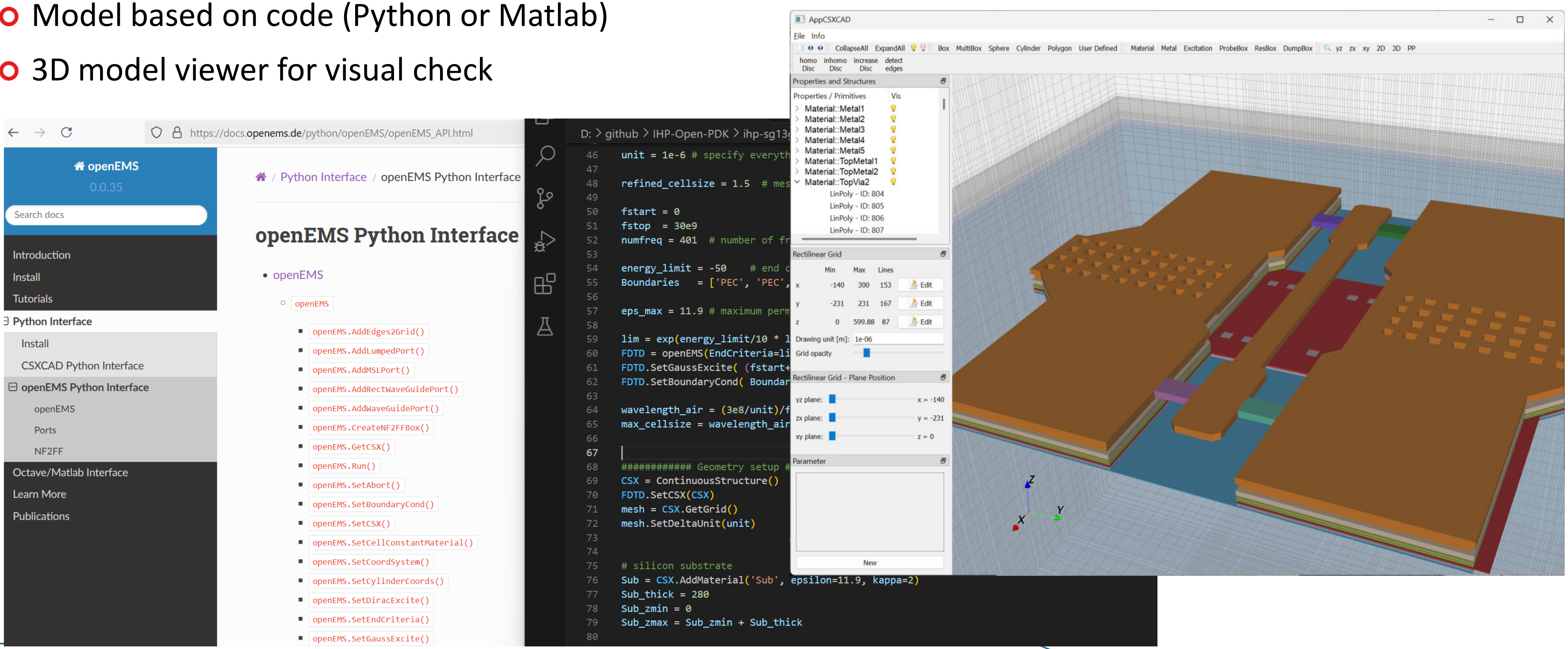
News and Anouncements

01.01.2023: Since the forum is still down, I have activated a github discussion to ask your question about the usage of openEMS or just give some feedback. Github Discussions

07.12.2022: New temporary openEMS.de website since the University DUE hosting is currently offline!

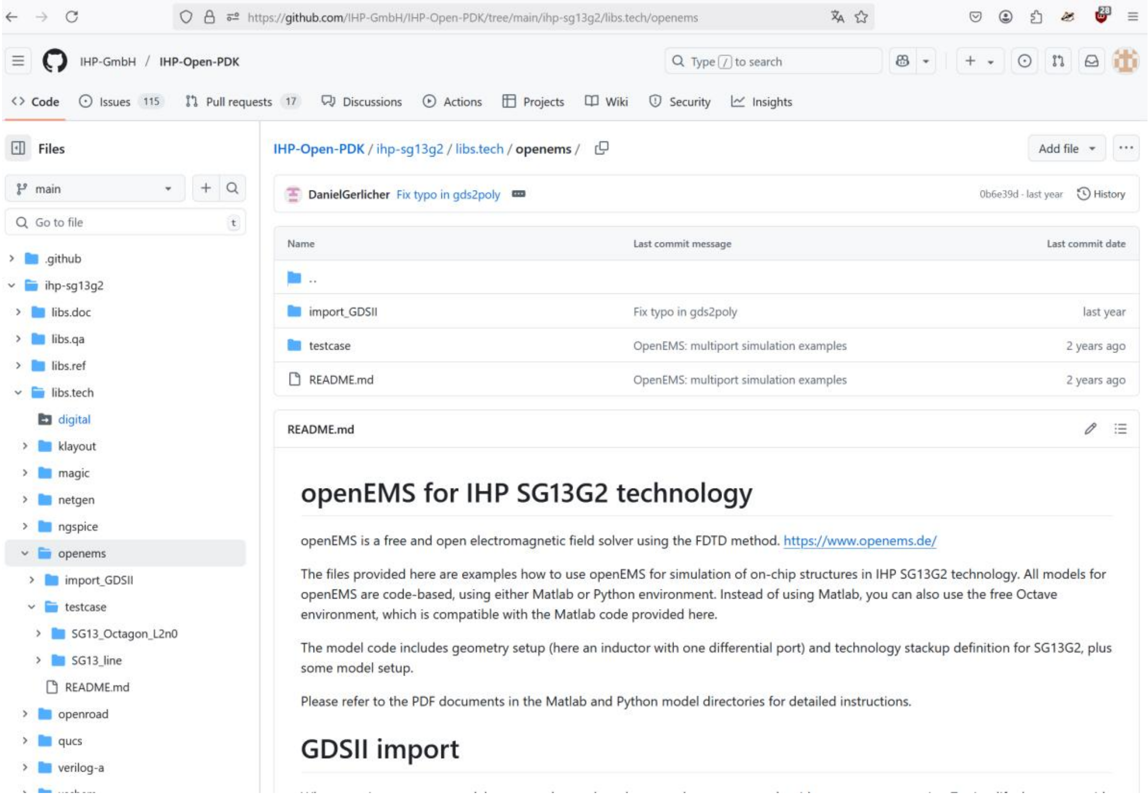
Basis of this work: openEMS

- Model based on code (Python or Matlab)
- 3D model viewer for visual check



Initial openEMS workflow in SG13G2 on github

- Initial version used standard openEMS code for modelling, proof of concept to verify accuracy
- This is NOT the new workflow discussed here!

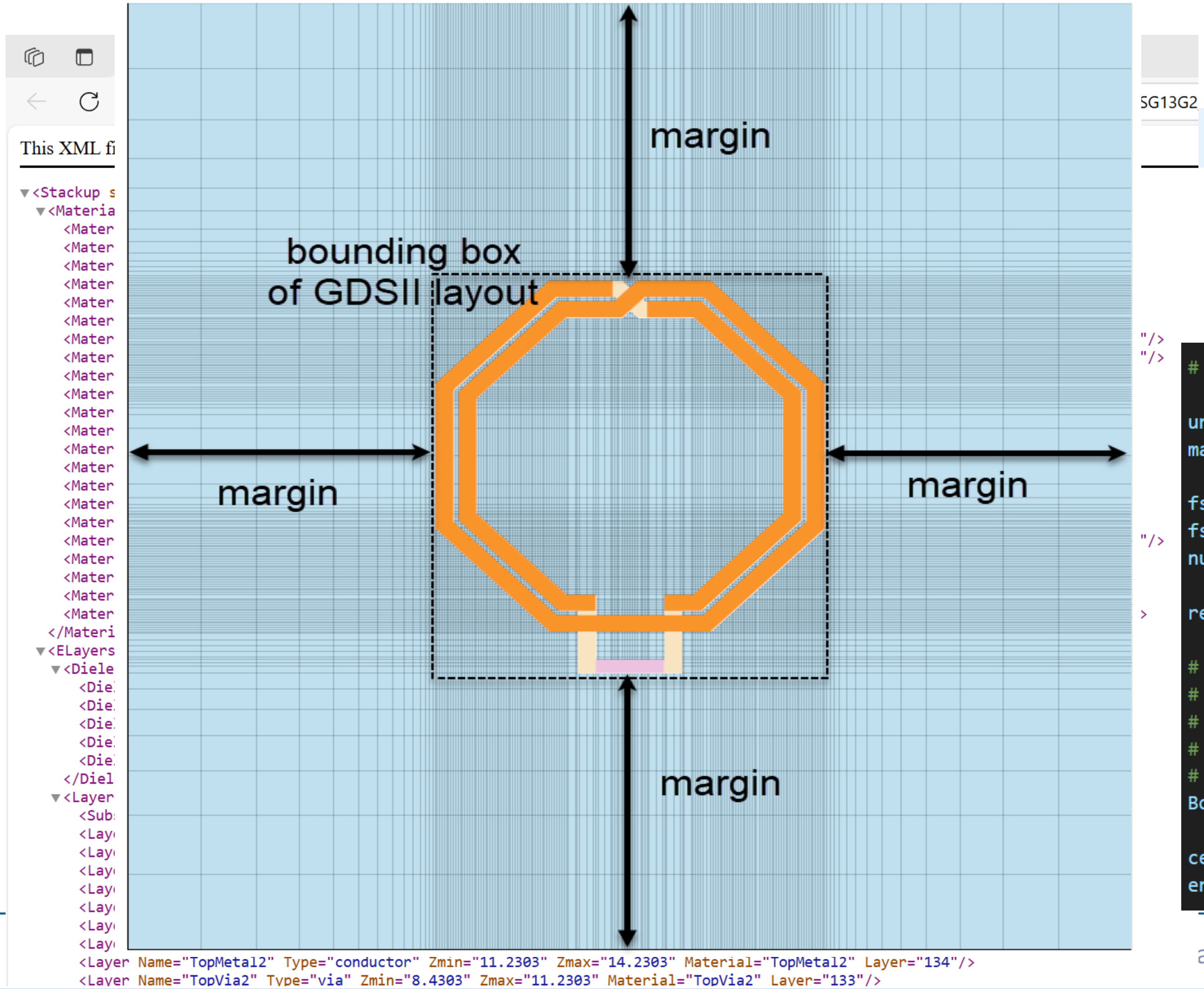


Make openEMS easier to use for RFIC work

- New workflow based on openEMS with Python API
- Additional software layer that specifically targets RFIC EM use cases
- Modular code, keep the main model simple & clean, easy to re-use
- Read geometries directly from GDSII layout files (no prior conversion needed)
- Read technology stackup from XML file
- Automatic meshing

- Idea: for new model, find the closest match from existing models and change only a few code lines

New workflow minimum example



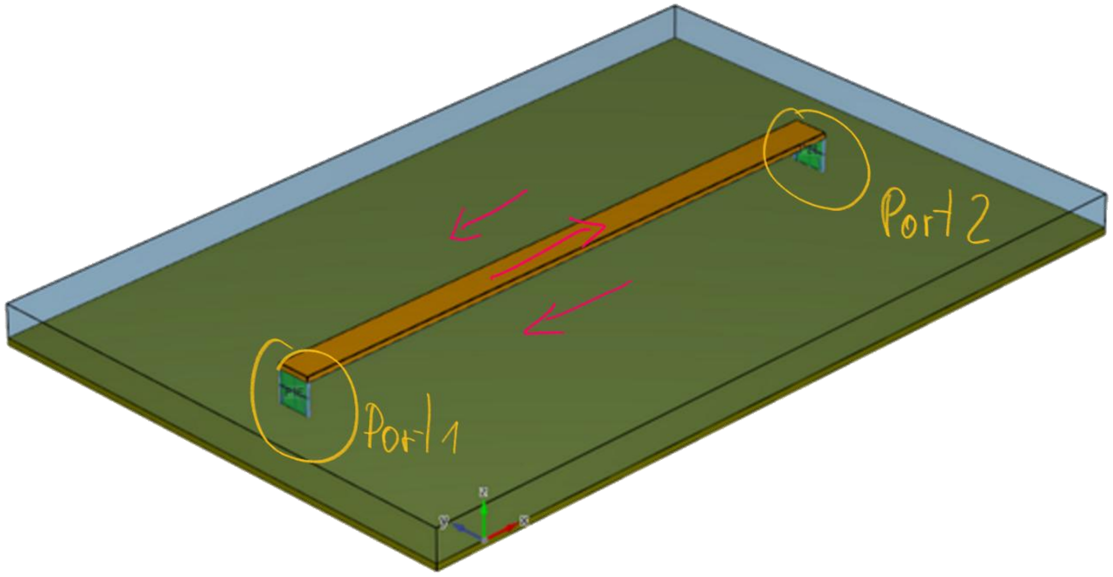
Utility function modules

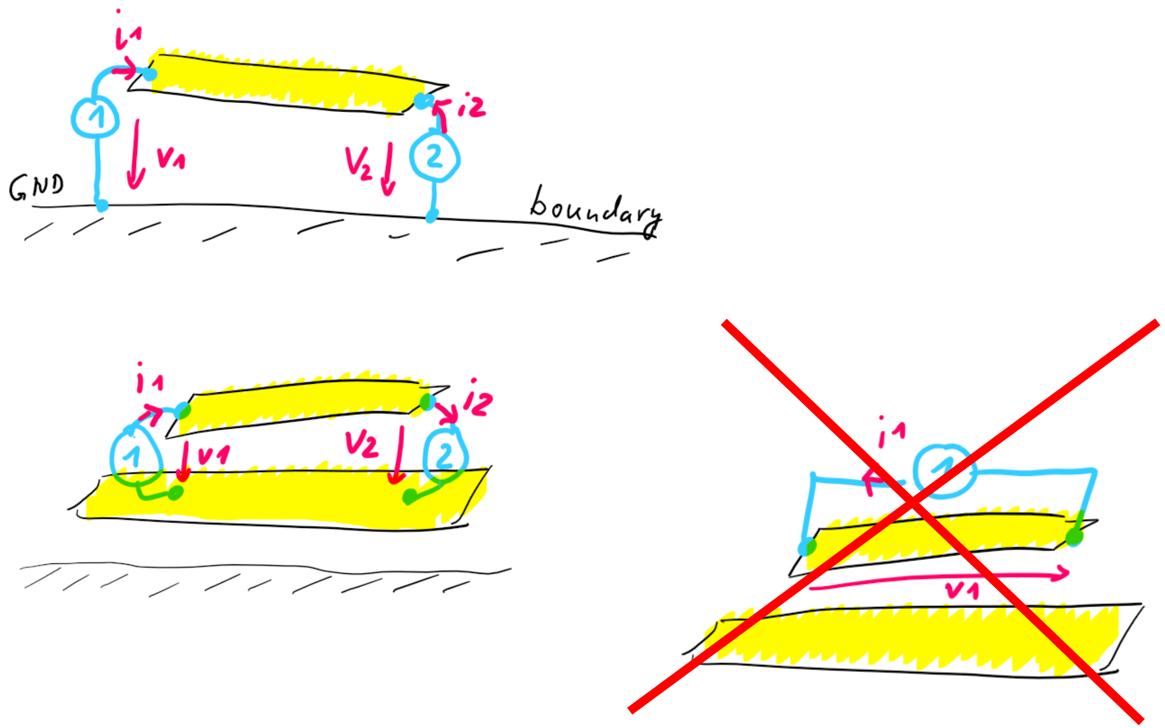
Technology stackup file

```
# ======================= 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 openEMS

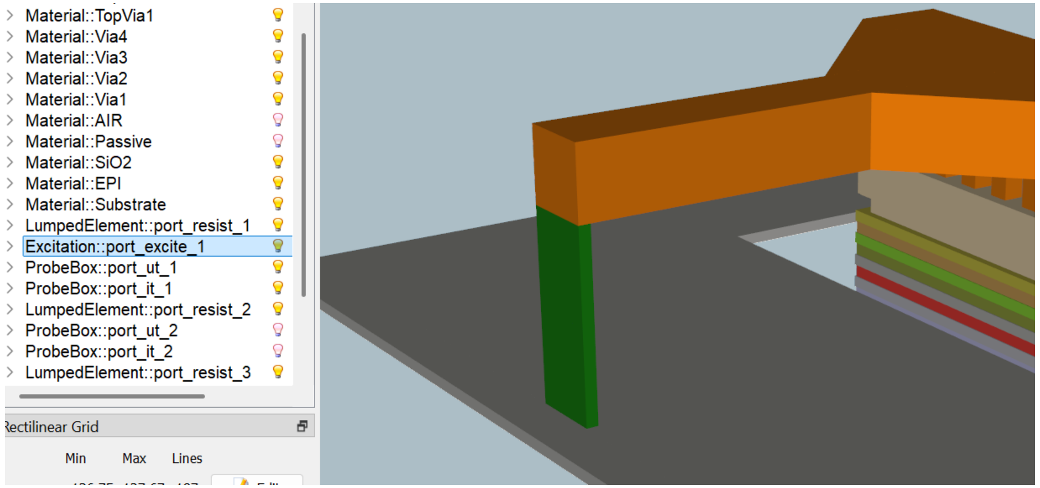
- Ports define the input/output interfaces to simulated layout
- Attention ADS Momentum users: ports require signal and reference terminal
- Rule: current only flows in closed loops!
- The port resistor physically closes the loop

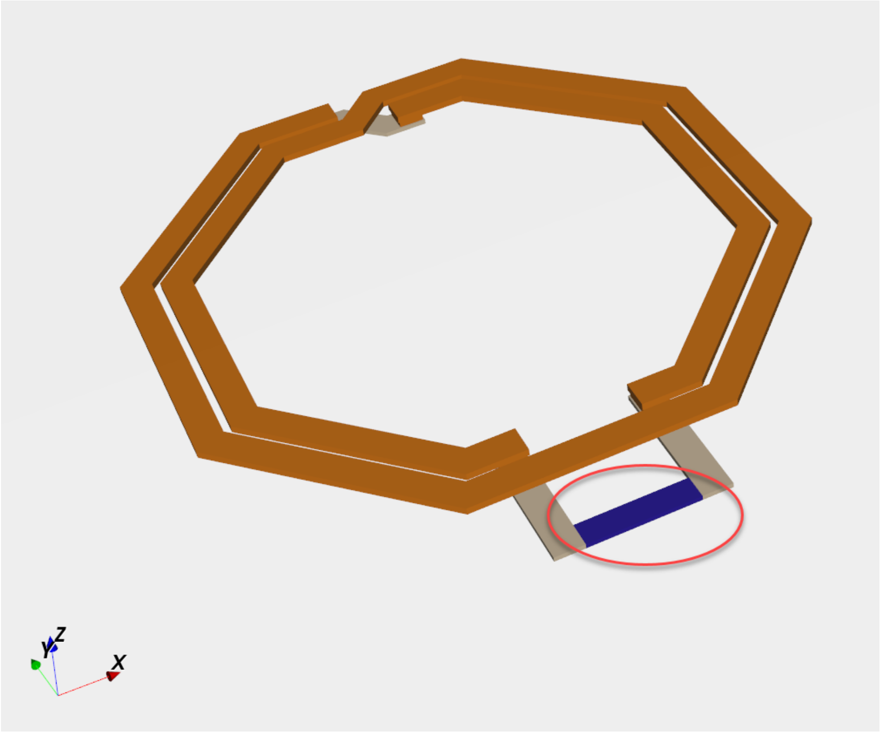




Ports: in-place or vertical between layers

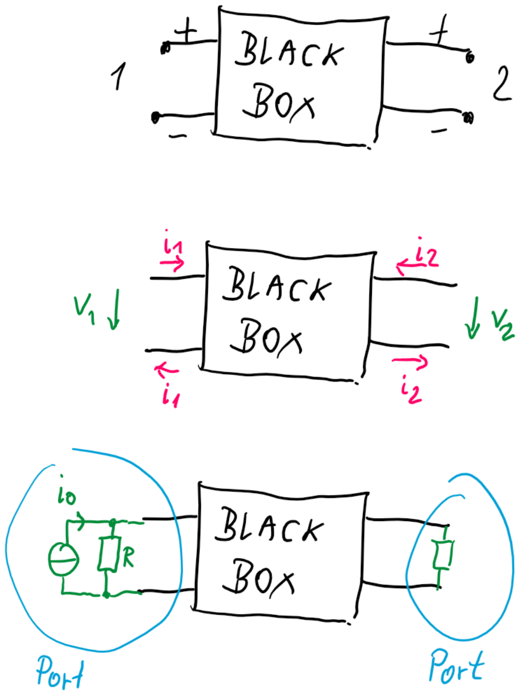
- In-plane ports between metals on same layer
- Vertical ports between metals of different layers
- Rule: current only flows in closed loops!

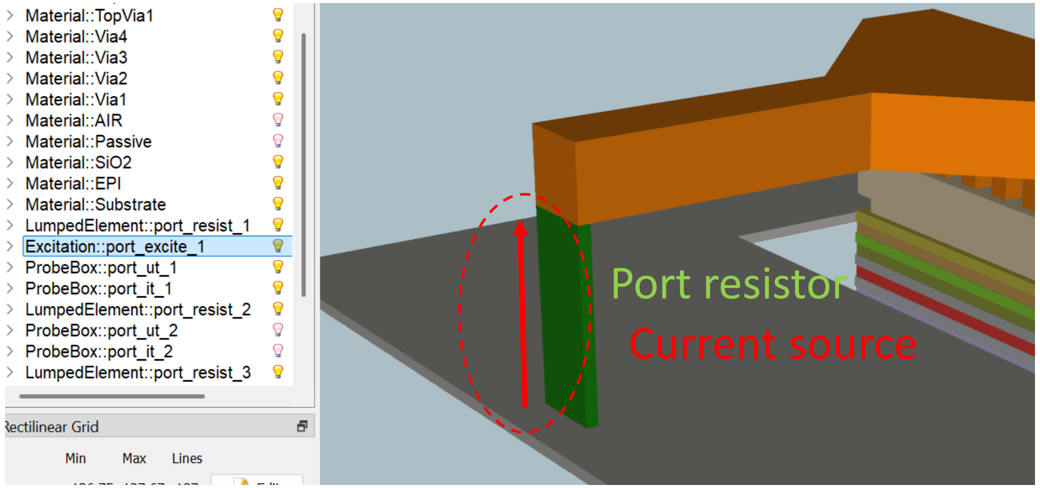


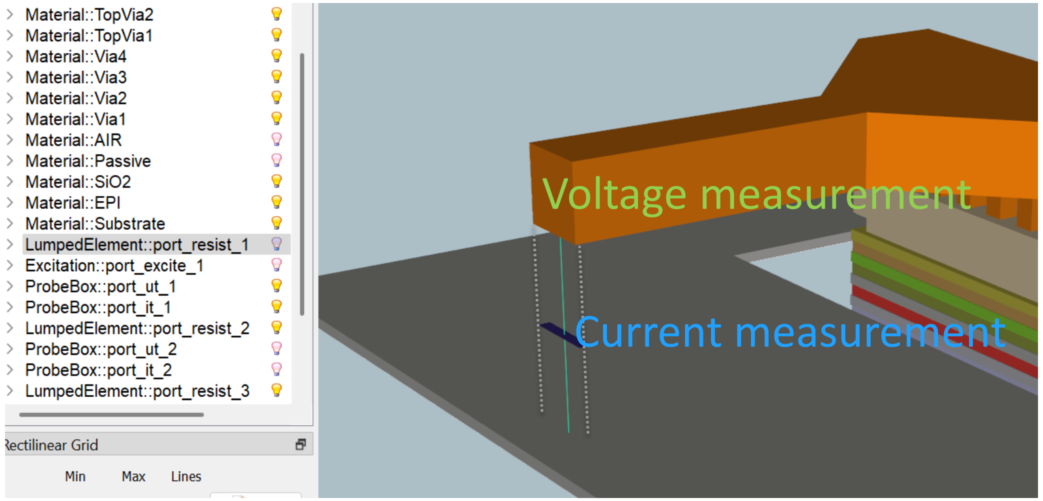


FDTD Method used in openEMS: Ports

- Ports can be in-plane or vertical
- Port is one line of code … but let's look at port internals here

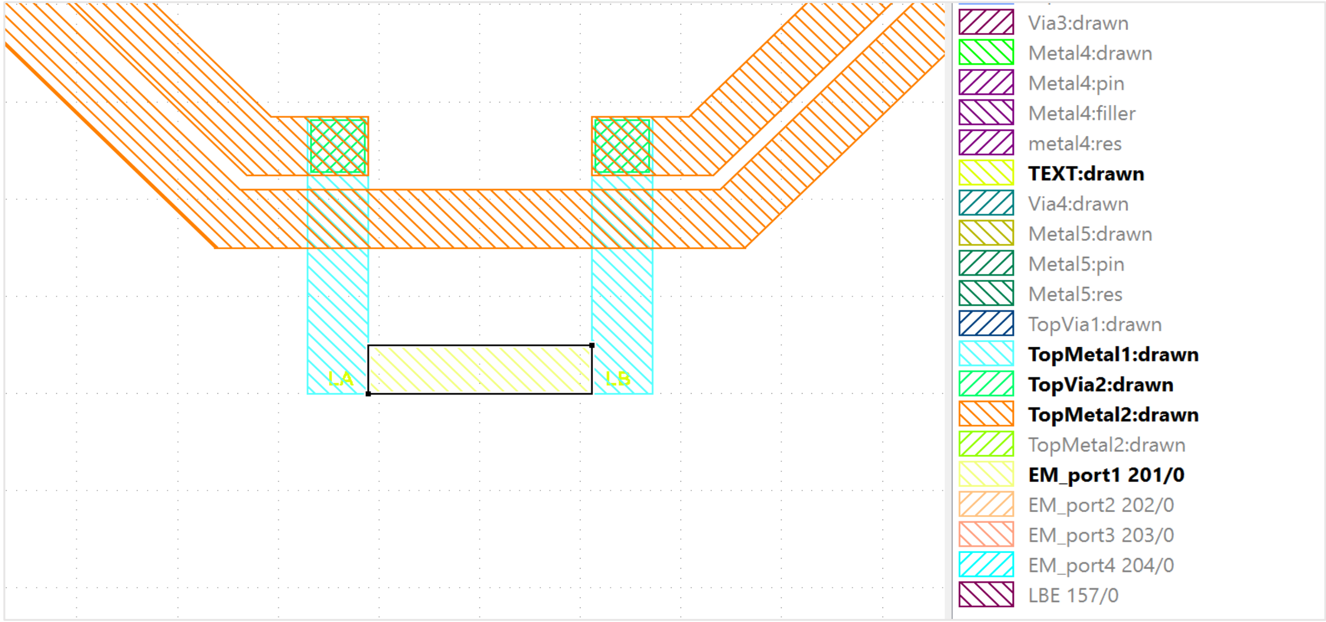






Port configuration in new workflow

- 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

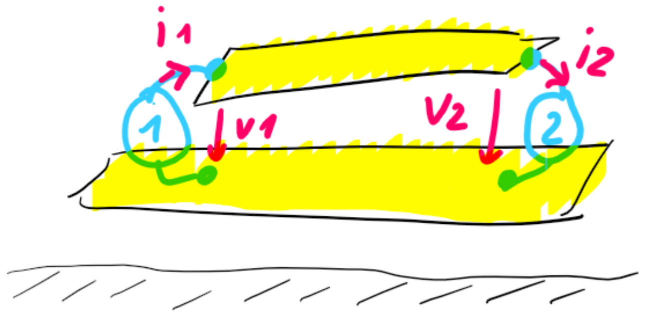



```
# ports from GDSII Data, polygon geometry from specified special layer
# note that for multiport simulation, excitations are switched on/off in simulation_setup.createSimulation below
simulation_ports = simulation_setup.all_simulation_ports()
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1,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

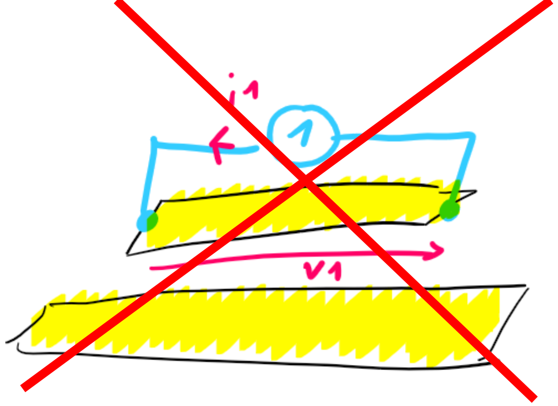
Why do we discuss ports in such detail?

- The EM simulator will always give you a result, even if your model setup is not what you intended to model
- Ports are one of the major error sources in EM simulation
- For accurate results, proper port setup is most important
- In openEMS and other 3D EM, keep ports small (compared to wavelength and other geometry)

valid



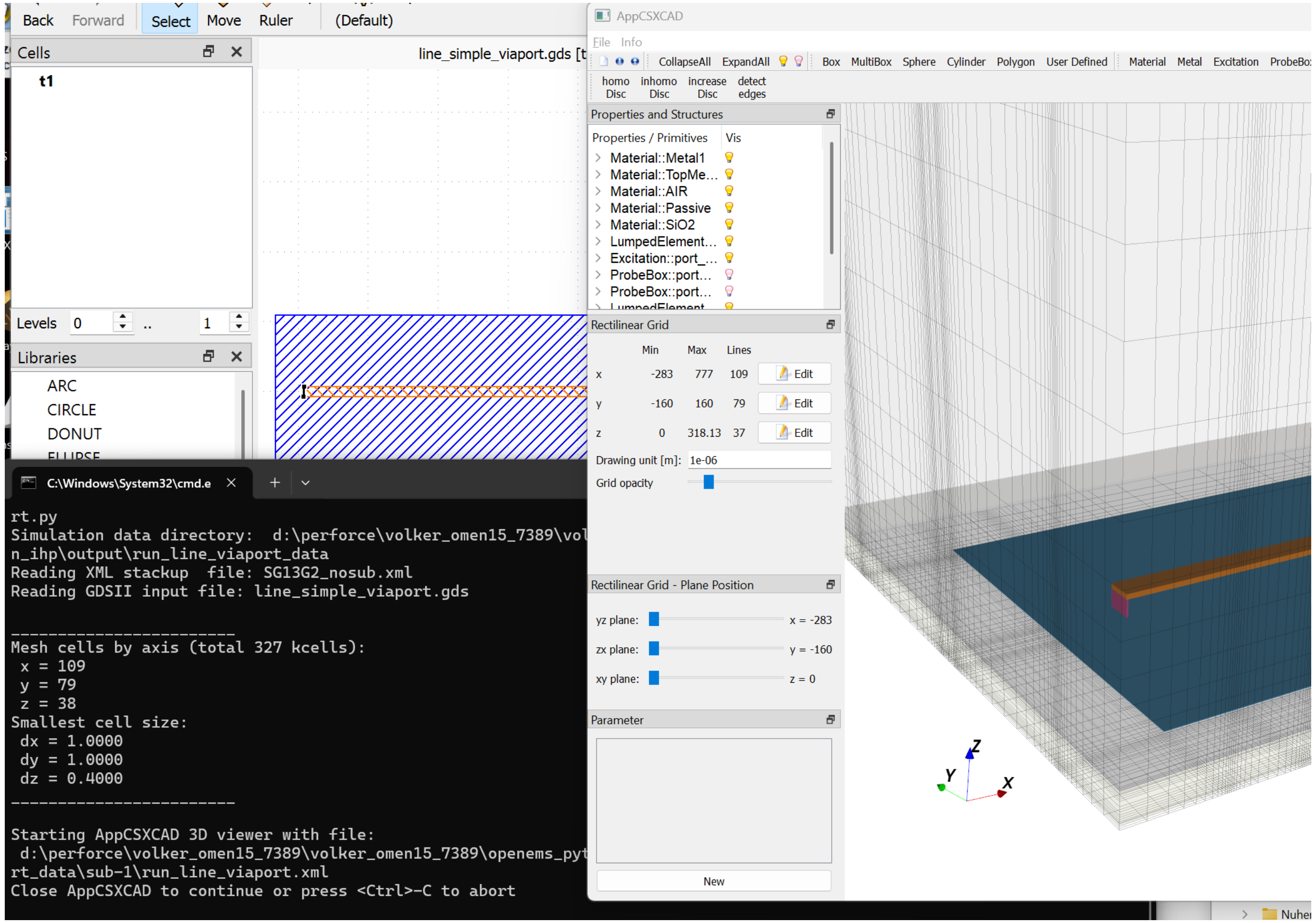
invalid large port



Overview

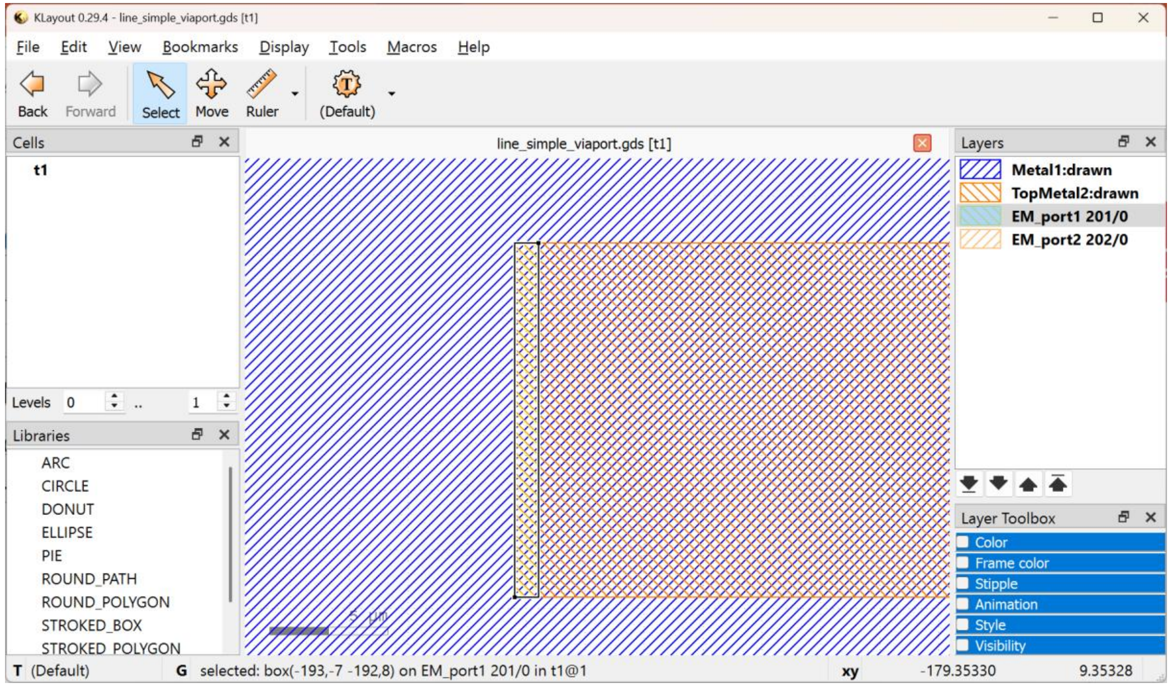
1. Why use EM simulation?
2. openEMS and FDTD method basics
3. What is different between standard openEMS and new Python workflow?
4. Examples transmission line and PA core
5. Simulation mesh
6. Using S-Parameter results, optional lumped model extraction
7. Summary

Simple straight line: run_line_viaport.py



GDSII file

- GDSII file has rectangles for TM2 transmission line and Metal1 ground return
- Ground polygon must be drawn for this simulation, only then we can include it in model
- Input port drawn as small box on layer 201
- Output port drawn as small box on layer 202



Code from run_line_viaport.py

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

# preview model/mesh only?
# postprocess existi
preview_only = False
postprocess_only = Fa

# ==================

gds_filename = "line_
XML_filename = "SG13G
```

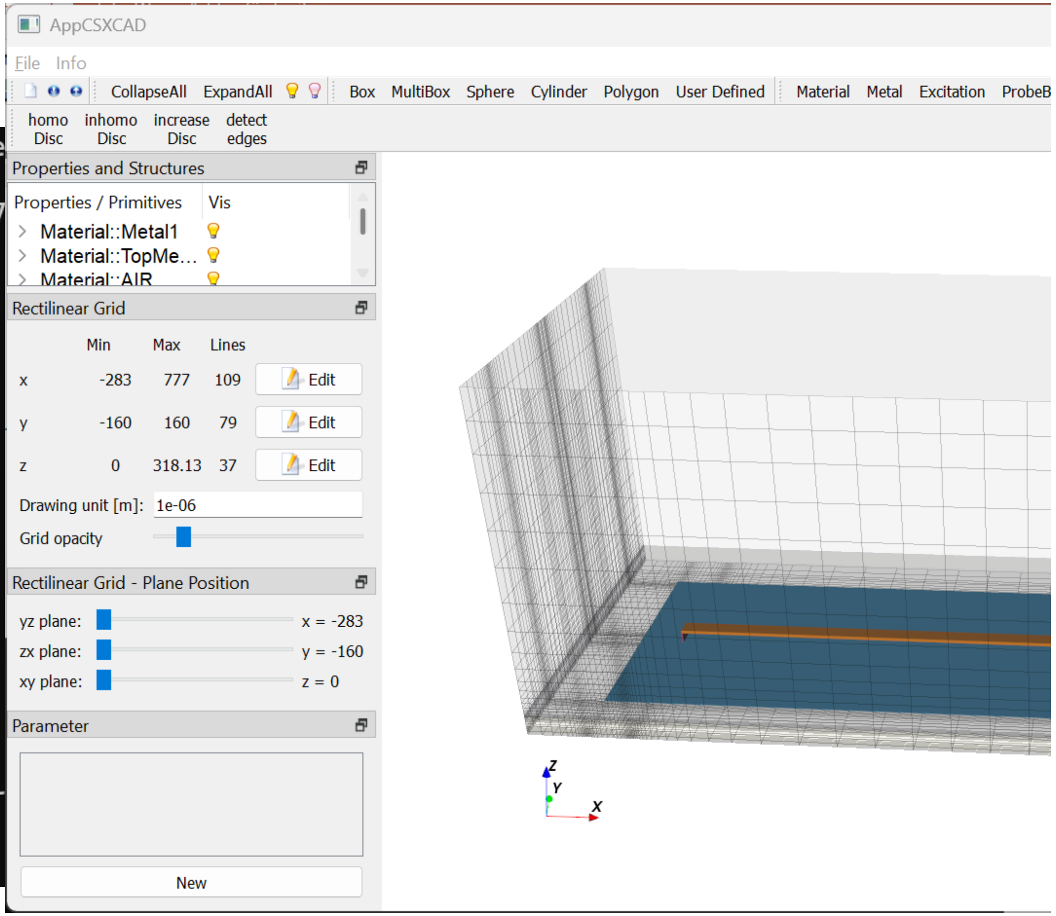
```
unit   = 1e-6  # geometry is in microns
margin = 50    # distance in microns from GDSII geometry boundary to simulation boundary

fstart =  0e9
fstop  = 110e9
numfreq = 401

refined_cellsize = 1  # mesh cell size in conductor region

# choices for boundary:
# 'PEC' : perfect electric conductor (default)
# 'PMC' : per
# 'MUR' : simp
# 'PML_8' : PM
Boundaries = [

cells_per_wave
energy_limit =
```

```
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

- Set model directory as current directory
- From command line: python3 run_line_viaport.py

```
D:\perforce\volker_OMEN15_7389\volker_OMEN15_7389\OpenEMS_Python_OPDK\Rele
t.py
Simulation data directory:  d:\perforce\volker_omen15_7389\volker_omen15_7
_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\rel
t_data\sub-1\run_line_viaport.xml
Close AppCSXCAD to continue or press <Ctrl>-C to abort
```



FDTD Simulation (running)

- To actually start simulation of this model, code must be `preview_only = False`
- Then, simulation starts after closing the 3D model viewer
- To exit without starting FDTD simulation, press Ctrl-C while 3D viewer is still open

```
C:\Windows\System32\cmd.e

        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)
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 simulation. Check your mesh!?
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) || Energy: ~5.31e-26 (- 0.00dB)
[@       10s] Timestep:          1995 || Speed:   62.3 MC/s (5.111e-03 s/TS) || Energy: ~4.47e-26 (- 0.74dB)
[@       15s] Timestep:          3325 || Speed:   92.0 MC/s (3.461e-03 s/TS) || Energy: ~1.07e-23 (- 0.00dB)
[@       20s] Timestep:          5320 || Speed:  120.1 MC/s (2.653e-03 s/TS) || Energy: ~6.52e-22 (- 0.00dB)
[@       25s] Timestep:          7315 || Speed:  139.5 MC/s (2.285e-03 s/TS) || Energy: ~8.49e-22 (- 0.00dB)
[@       30s] Timestep:          9975 || Speed:  161.0 MC/s (1.979e-03 s/TS) || Energy: ~1.97e-19 (- 0.00dB)
[@       35s] Timestep:         12635 || Speed:  177.5 MC/s (1.795e-03 s/TS) || Energy: ~3.09e-19 (- 0.00dB)
[@       39s] Timestep:         15253 || Speed:  187.7 MC/s (1.697e-03 s/TS) || Energy: ~2.19e-18 (- 0.00dB)
```

FDTD Simulation (finished)

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

```
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)
```

```
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 simulation. Ch
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) || Energy: ~
[@       10s] Timestep:         1995 || Speed:   62.3 MC/s (5.111e-03 s/TS) || Energy: ~
[@       15s] Timestep:         3325 || Speed:   92.0 MC/s (3.461e-03 s/TS) || Energy: ~
[@       20s] Timestep:         5320 || Speed:  120.1 MC/s (2.653e-03 s/TS) || Energy: ~
[@       25s] Timestep:         7315 || Speed:  139.5 MC/s (2.285e-03 s/TS) || Energy: ~
[@       30s] Timestep:         9975 || Speed:  161.0 MC/s (1.979e-03 s/TS) || Energy: ~
[@       35s] Timestep:        12635 || Speed:  177.5 MC/s (1.795e-03 s/TS) || Energy: ~
[@       39s] Timestep:        15253 || Speed:  187.7 MC/s (1.697e-03 s/TS) || Energy: ~2.19e-18 (- 0.00dB)
[@       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
```

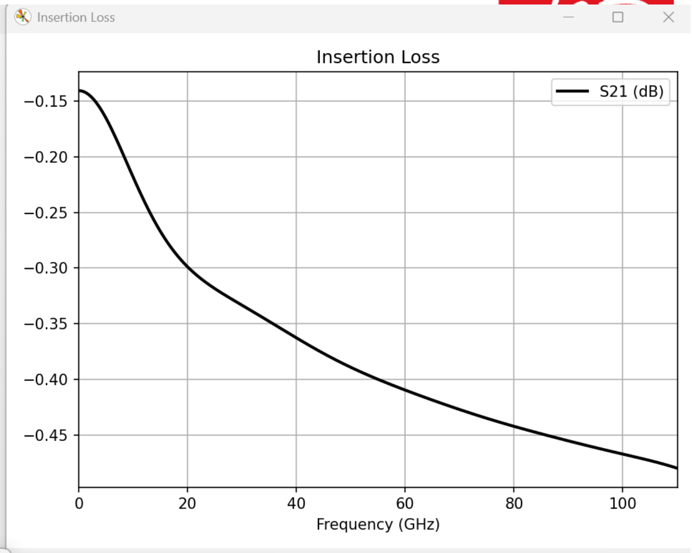
Data plot

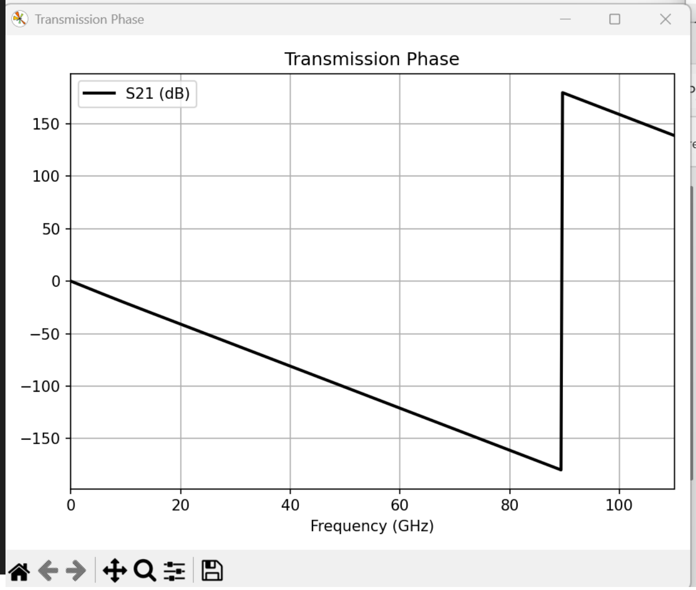
```python
sub1_data_path = simulation_setup.runSimulation (excite_ports, FDTD, sim_path, model_basename,


########## 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()
```

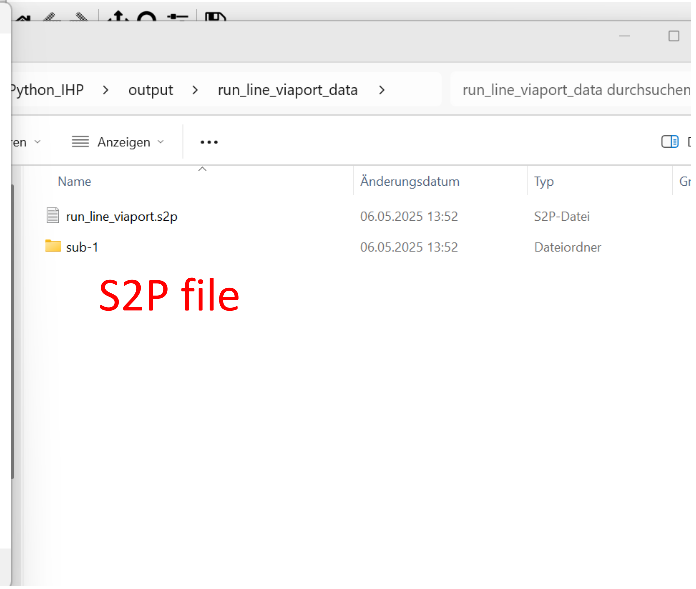
fake symmetry

write S2P file

S11 plot

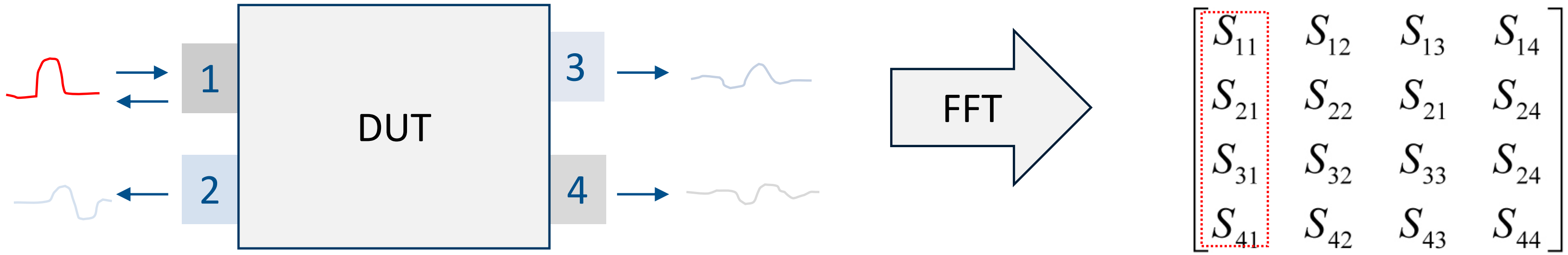






Port excitations (again)

- Remember that one port excitation does not give us the full [S] matrix



- For symmetric DUT we can assume S11=S22 and S12=S21
- For general case, our model code can run multiple port excitations, one after another

```
# Create simulation for port 1 .. 4 excitation, return value is data path for that excitation

excite_ports_list = [[1],[2],[3],[4]]  # list of ports that are excited in the different excitation runs
for excite_ports in excite_ports_list:
    # define excitation and stop criteria and boundaries
    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, dielect
    simulation_setup.runSimulation (excite_ports, FDTD, sim_path, model_basename, preview_only, postproces
```

Model data structure in file system

- Each excitation creates a separate directory to store data
- Combined S-parameter file stored one level above

sub-1	11.03.2025 16:18	Dateiordner
sub-2	11.03.2025 16:45	Dateiordner
sub-3	11.03.2025 17:16	Dateiordner
sub-4	11.03.2025 20:29	Dateiordner
run_core_50ghz_mpa.s4p	11.03.2025 20:58	S4P-Datei

output › run_core_50ghz_mpa_data › sub-1

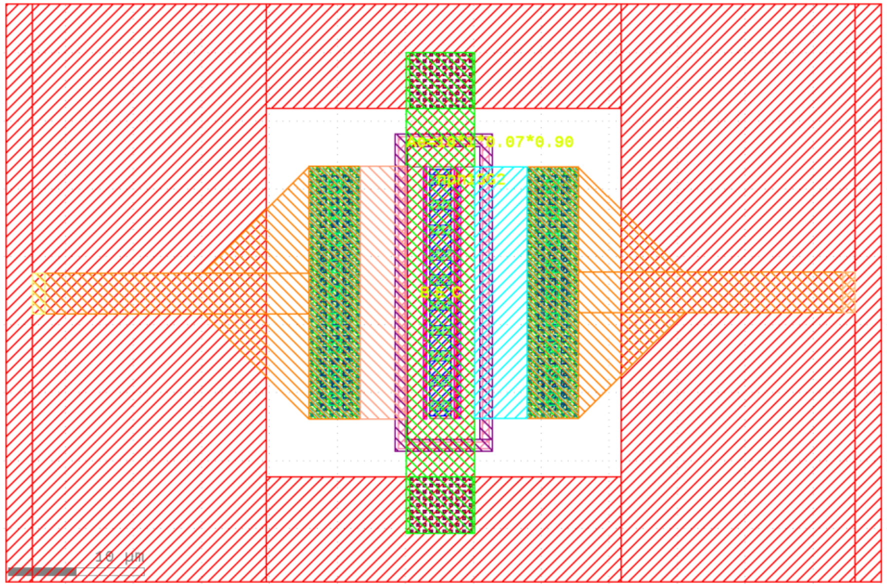
Anzeigen

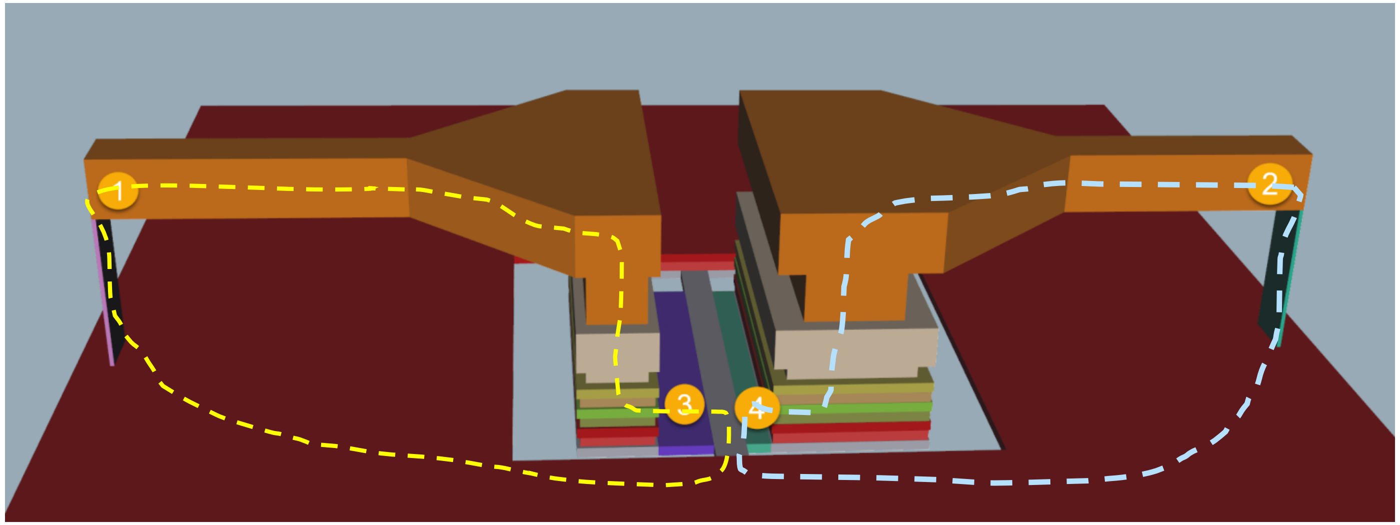
Name	Änderungsdatum
et	11.03.2025 18:56
ht	11.03.2025 18:56
port_it_1	11.03.2025 19:26
port_it_2	11.03.2025 19:26
port_it_3	11.03.2025 19:26
port_it_4	11.03.2025 19:26
port_ut_1	11.03.2025 19:26
port_ut_2	11.03.2025 19:26
port_ut_3	11.03.2025 19:26
port_ut_4	11.03.2025 19:26
run_core_50ghz_mpa.xml	11.03.2025 18:55

```
# get results, CSX port definition is read from simulation ports object
# all S-parameter data is available because we have simulated all port excitations
s11 = utilities.calculate_Sij (1, 1, f, sim_path, simulation_ports)
s21 = utilities.calculate_Sij (2, 1, f, sim_path, simulation_ports)
s31 = utilities.calculate_Sij (3, 1, f, sim_path, simulation_ports)
s41 = utilities.calculate_Sij (4, 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)
s32 = utilities.calculate_Sij (3, 2, f, sim_path, simulation_ports)
s42 = utilities.calculate_Sij (4, 2, f, sim_path, simulation_ports)
s13 = utilities.calculate_Sij (1, 3, f, sim_path, simulation_ports)
s23 = utilities.calculate_Sij (2, 3, f, sim_path, simulation_ports)
s33 = utilities.calculate_Sij (3, 3, f, sim_path, simulation_ports)
s43 = utilities.calculate_Sij (4, 3, f, sim_path, simulation_ports)
s14 = utilities.calculate_Sij (1, 4, f, sim_path, simulation_ports)
s24 = utilities.calculate_Sij (2, 4, f, sim_path, simulation_ports)
s34 = utilities.calculate_Sij (3, 4, f, sim_path, simulation_ports)
s44 = utilities.calculate_Sij (4, 4, f, sim_path, simulation_ports)

s4p_name = os.path.join(sim_path, model_basename + '.s4p')
utilities.write_snp (np.array([[s11,s21,s31,s41], [s12,s22,s32,s42], [s13,s23,s33,s43], [s14,s24,s34,s44]]),f, s4p_name)
```

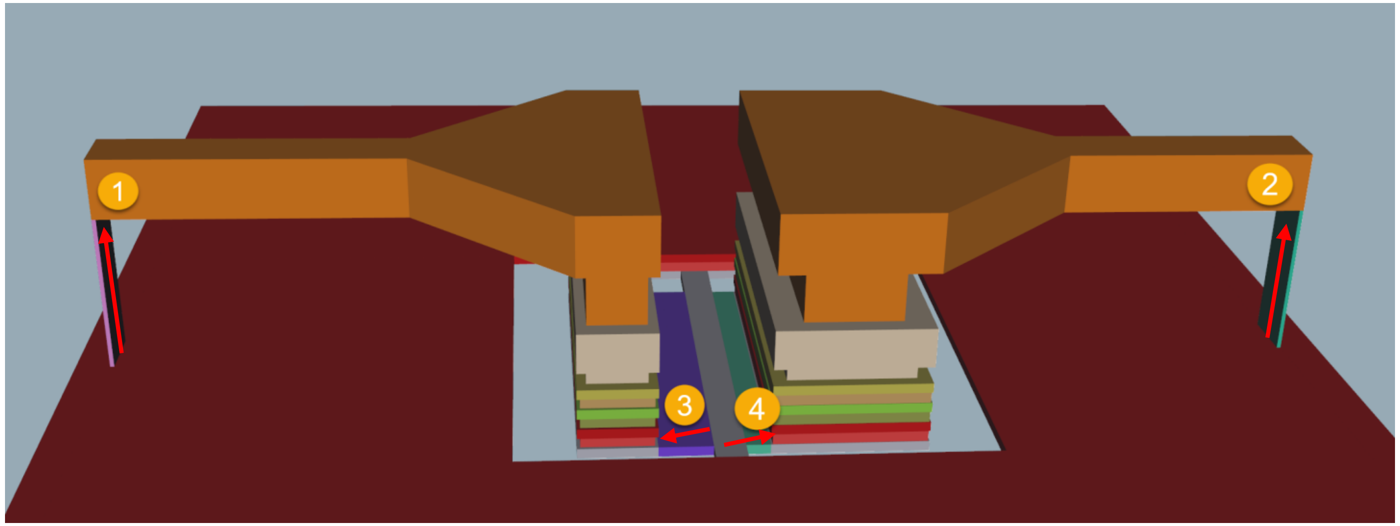
Example model PA core

- PA core is layout with transistor feed + via stack, transistor itself is modelled by ports
- Transistor has center „strip“ as common terminal for Emitter
- Port 3 (Base) and Port 4 (Collector) use that shared ground reference
- Note the closed loops for currents





PA core port definitions



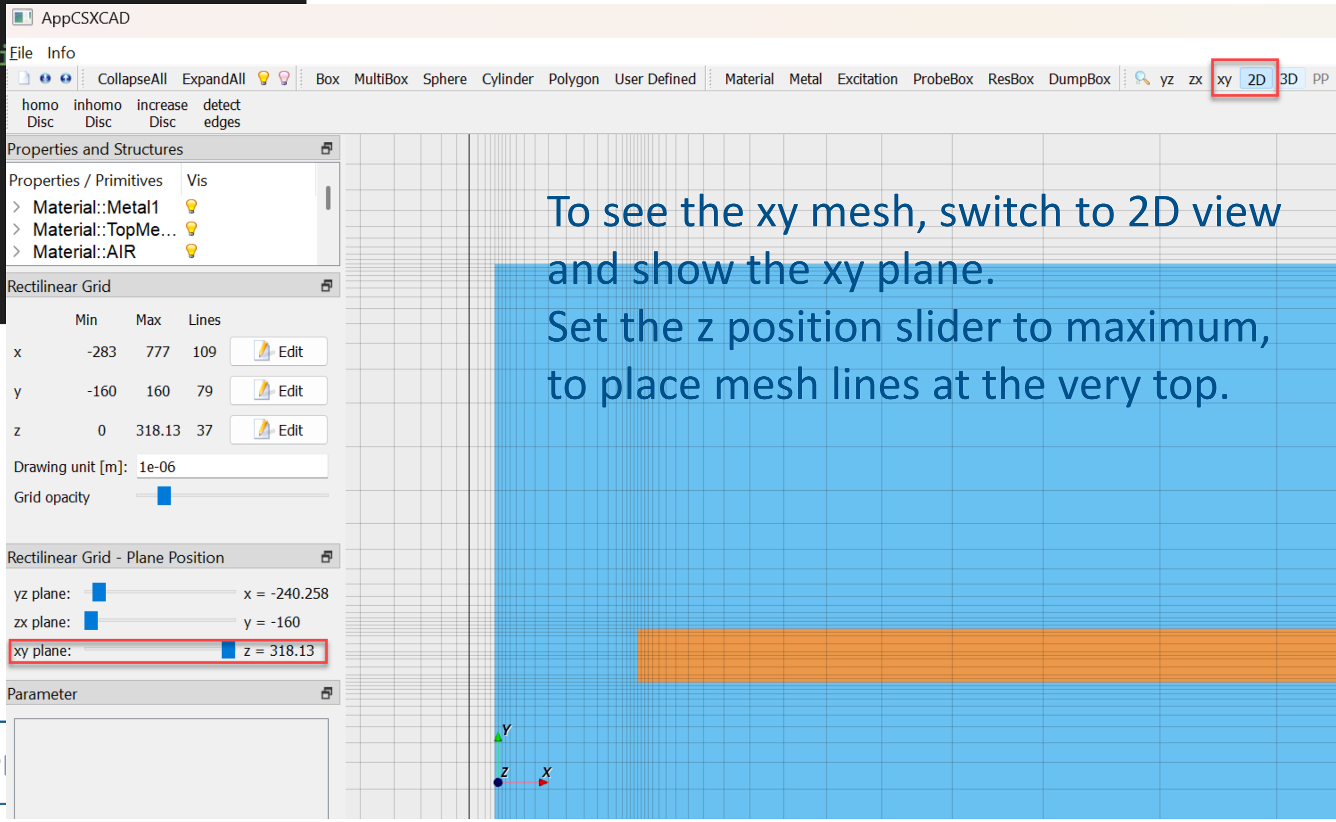

```python
simulation_ports = simulation_setup.all_simulation_ports()
# instead of in-plane port specified with target_layername, we here use via port specified with from_layername and to_layername.
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=1, voltage=1, port_Z0=50, source_layernum=201,
                                                           from_layername='Metal3', 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='Metal3', to_layername='TopMetal2', direction='z'))
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=3, voltage=1, port_Z0=50, source_layernum=203,
                                                           target_layername='Metal2', direction='-x'))
simulation_ports.add_port(simulation_setup.simulation_port(portnumber=4, voltage=1, port_Z0=50, source_layernum=204,
                                                           target_layername='Metal2', direction='x'))
```

Overview

Simulation Mesh

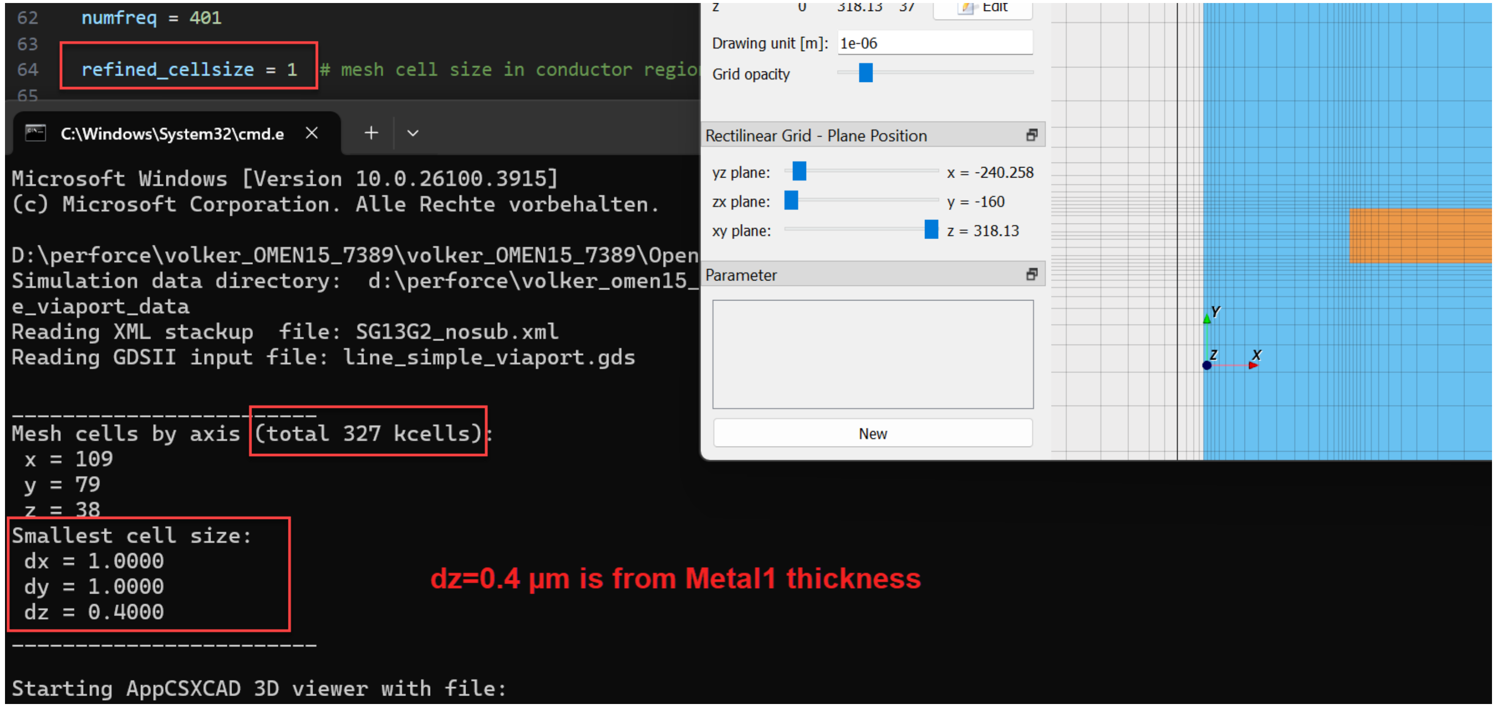
- Creating the simulation mesh is a major difficulty in standard openEMS
- In this new workflow for IHP SG13G2, mesh is built automatrically based on geometry
- One user input to set required minimum mesh size (required detail resolution)

```
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
```



Why not always use very fine mesh?

- Using small cell size has two effects:
  - Minimum mesh size gets smaller => simulation time step gets smaller => more time steps needed
  - More mesh cells required for simulation volume
  - These two effects combined: solve many more cells for many more timesteps => longer simul. time



Simulation time with different mesh size

- `refined_cellsize = 1:   min=0.40, 327 kcells, 37240 timesteps, 1min:14s`
- `refined_cellsize = 0.5: min=0.40, 582 kcells, 61957 timesteps, 2min:35s`
- `refined_cellsize = 0.2: min=0.14, 1439 kcells, 212408 timesteps, 29min:41s`

- When Metal1 with thickness 0.4 µm is minimum mesh dimension, we simply see the difference in total mesh count, the FDTD timestep is unchanged
- When refined_cellsite creates mesh cell smaller than that, we have the combined effect of more mesh cells and smaller mesh cell => smaller time step, so total simulation time is much longer

```
________________________
Mesh cells by axis (total 1460 kcells):
 x = 163
 y = 128
 z = 70
Smallest cell size:
 dx = 0.2000
 dy = 0.1796
 dz = 0.1400
________________________
```

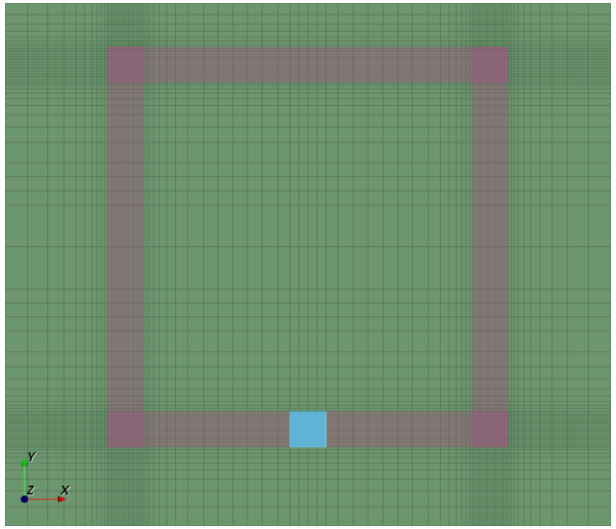
```
Create FDTD operator (compressed SSE + multi-threading)
FDTD simulation size: 163x128x69 --> 1.43962e+06 FDTD cells
FDTD timestep is: 2.9952e-16 s; Nyquist rate: 15175 timesteps @1.10006e+11 Hz
openEMS::SetupFDTD: Warning, the timestep seems to be very small --> long simulation. Check your mesh!?
Excitation signal length is: 173902 timesteps (5.20872e-11s)
Max. number of timesteps: 1000000000 ( --> 5750.37 * Excitation signal length)
Create FDTD engine (compressed SSE + multi-threading)
Running FDTD engine... this may take a while... grab a cup of coffee?!?
```

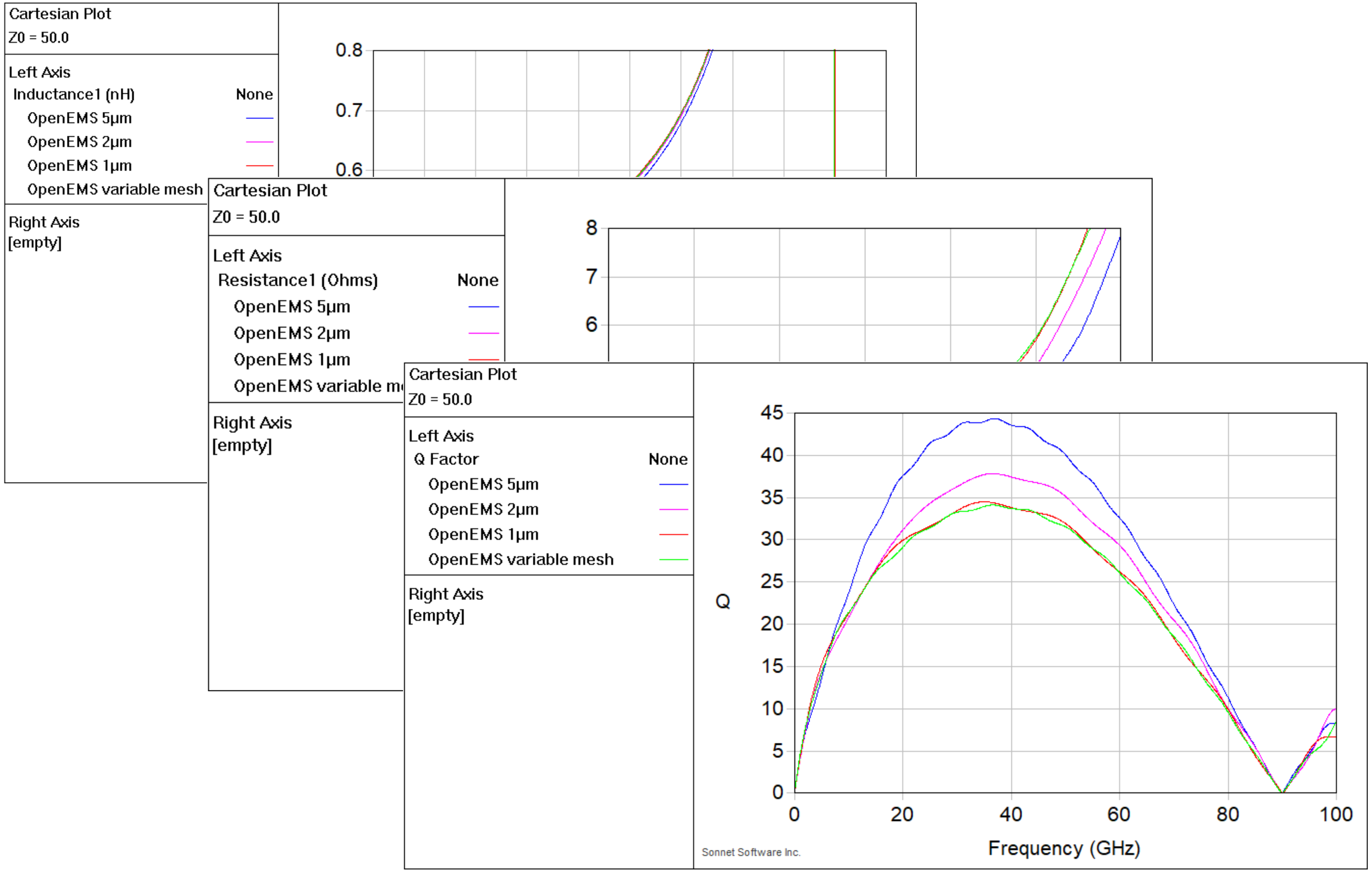
Simulation Mesh and Skin Effect

- Conductors modelled as solid volumes with conductivity
- Must mesh into skin depth to include skin effect in results
- Images below from another EM solver, only to show effect
- Skin depth for Aluminium:
  - 0.58µm @ 20 GHz
  - 0.37µm @ 50 GHz
  - 0.26 µm @ 100 GHz
- Set refined_cellsize with skin effect in mind, if loss matters to your model

$$\delta = \sqrt{\frac{\rho}{\pi f_o \mu_r \mu_o}}$$

Simple square loop at different mesh size





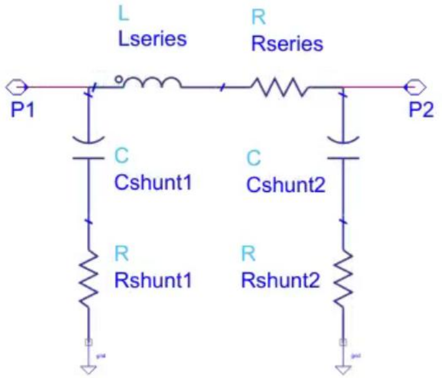
Mesh resolution summary

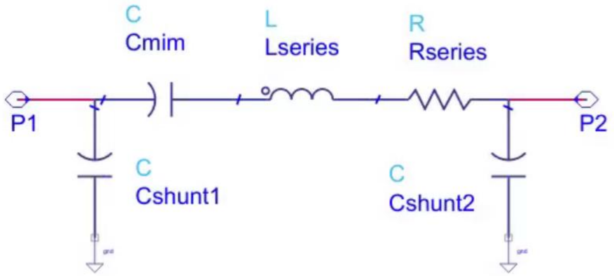
- Mesh resolution must resolve small geometry detail
- Mesh resolution matters to get correct skin effect loss
- Set mesh resolution as small as necessary, but not smaller ;-)
- Learn from running a few test cases (convergence study)

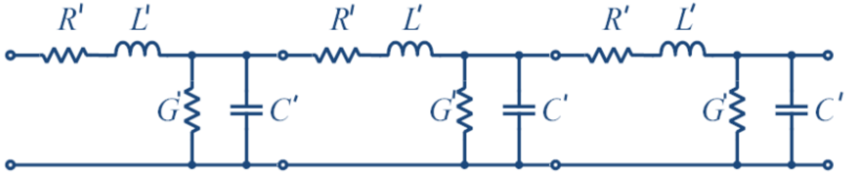
Overview

Using results

- S-parameter output can be used in circuit simulation
- No noise simulation in ngspice with S-parameter data -> workaround is lumped circuit model extraction
- Lumped model extraction also useful for transmission line models
- Lumped model ensure we have no leakage currents from small numerical errors
- Downloads: https://github.com/VolkerMuehlhaus/lumpedmodel







Overview

Summary

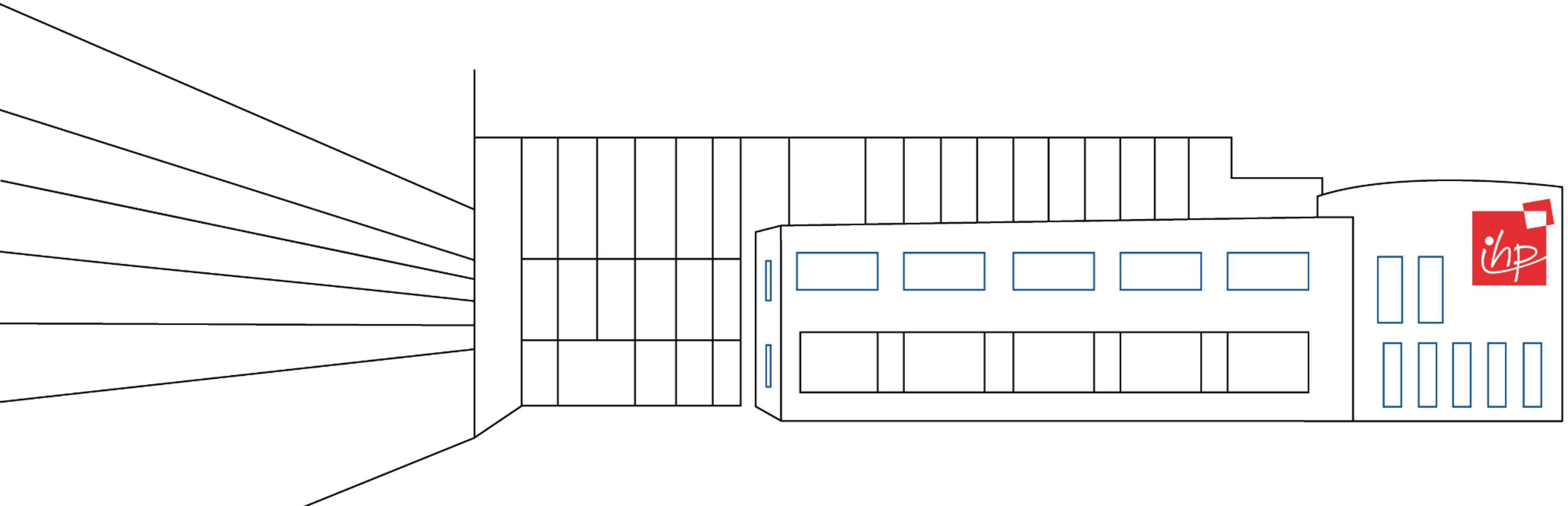
- openEMS is time domain EM simulation, results are the usual *.snp Touchstone files
- New workflow takes GDSII input files, plus some simulation settings, in one Python model file
- Simulation ports must be added to GDSII file on extra layers (recommended 201 and above)
- Port direction is set in Python model file, pay attention to port polarity (otherwise 180° phase shift at DC)
- Simulation mesh requires one user input, to set mesh resolution (as fine as necessary)
- Multiple ports require excitation from one port after another, all controlled from a single Python model file

References

- openEMS:
  https://www.openems.de/
- New IHP workflow for GDSII files
  https://github.com/VolkerMuehlhaus/openems_ihp_sg13g2
- Lumped model extraction:
  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 – Innovations for High Performance Microelectronics
Im Technologiepark 25
15236 Frankfurt (Oder)
Tel.: +49 (0) 335 5625
E-Mail: volker@muehlhaus.com



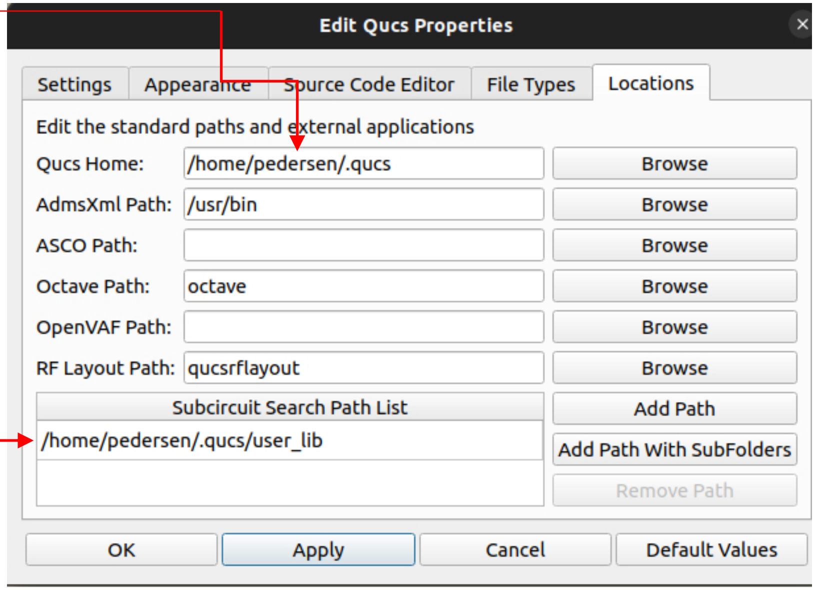
Leibniz
Association

Setting Up Qucs-s

- **Navigate to the following directory:** IHP-Open-PDK/ihp-sg13g2/libs.tech/qucs
- **Run the following:** *python3 install.py*
- **Launch Qucs by writing the following in the command line:** *qucs-s*
- **Navigate to the following location in Qucs:** *file -> Application settings -> Locations tab*

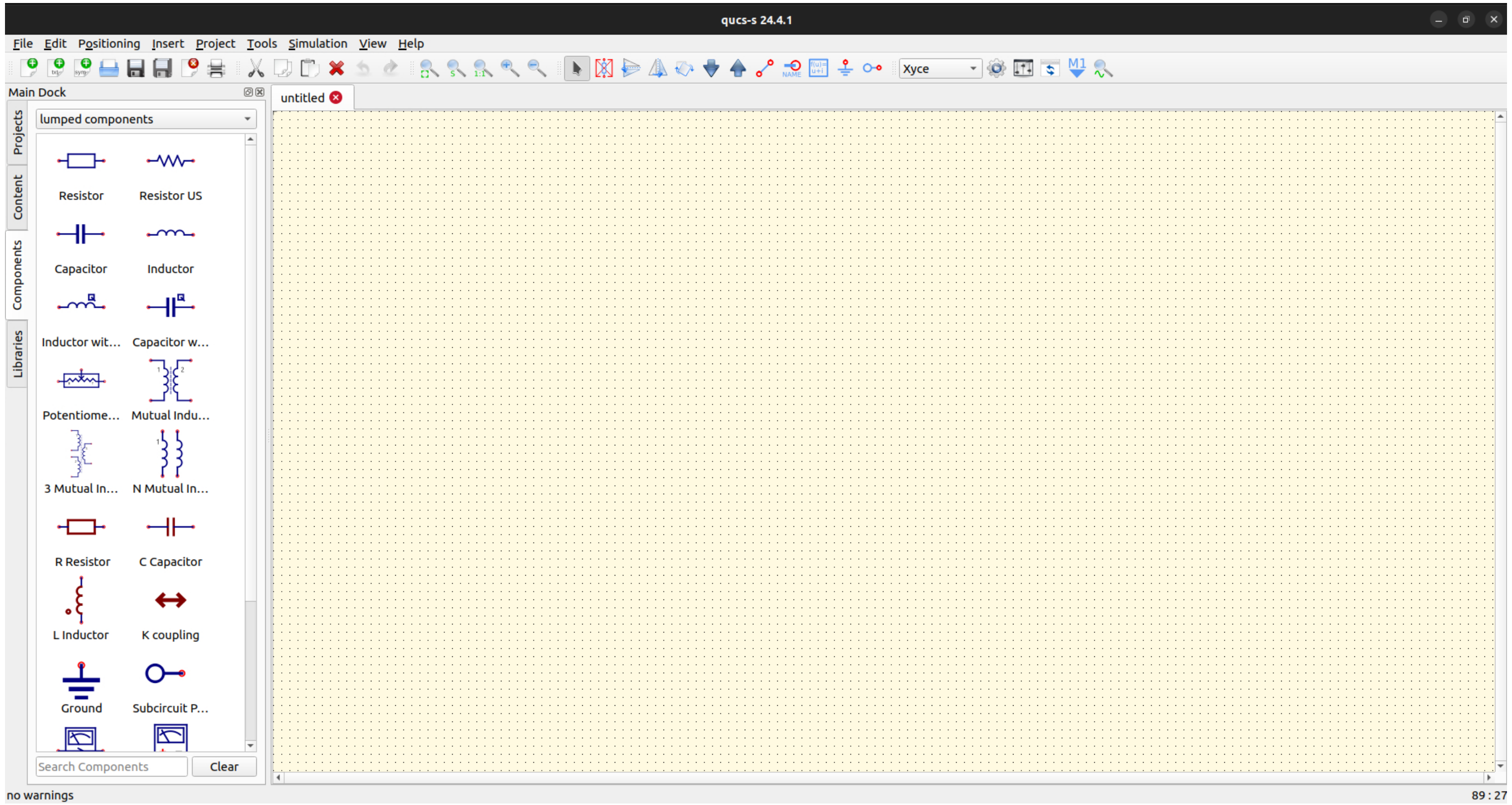
Change to your own path!

Press "Ctrl+H" to show hidden folders



Simple Navigation in QUCS-S

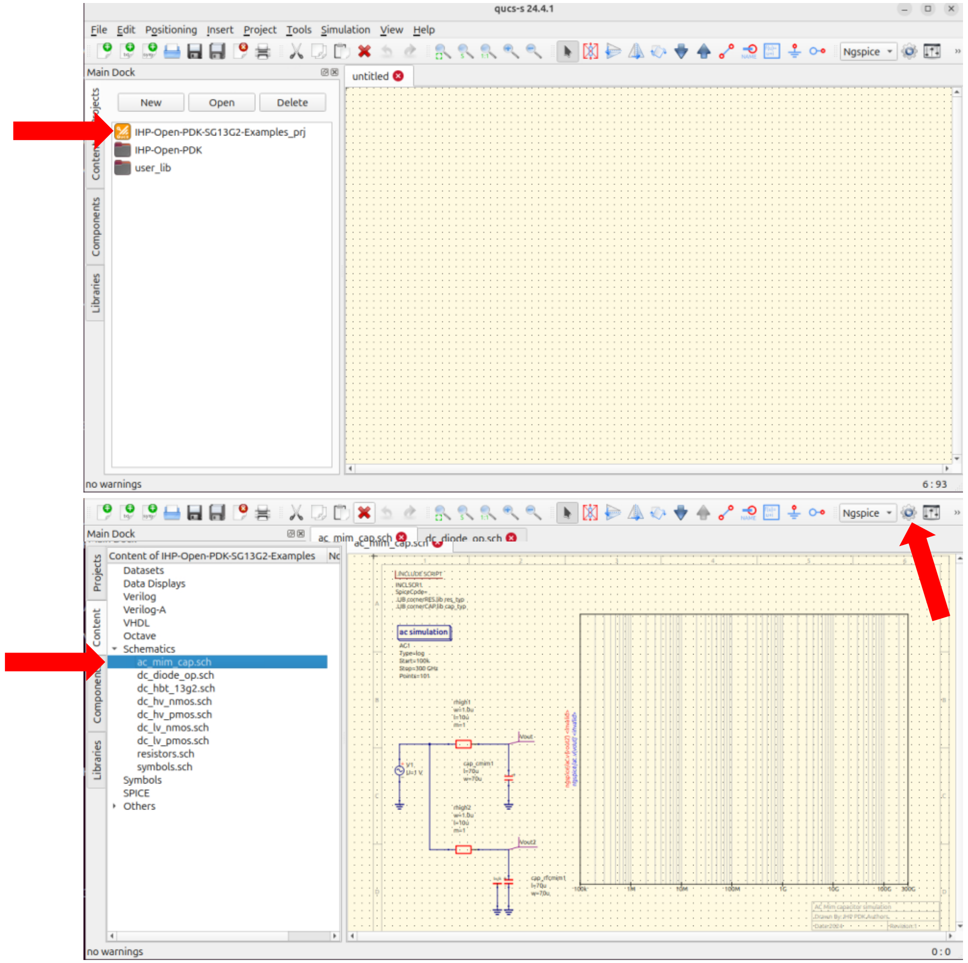
- **To launch without specific file reference:** *qucs-s*
- **Opening a specific file:** *qucs-s –i path/to/file.sch*

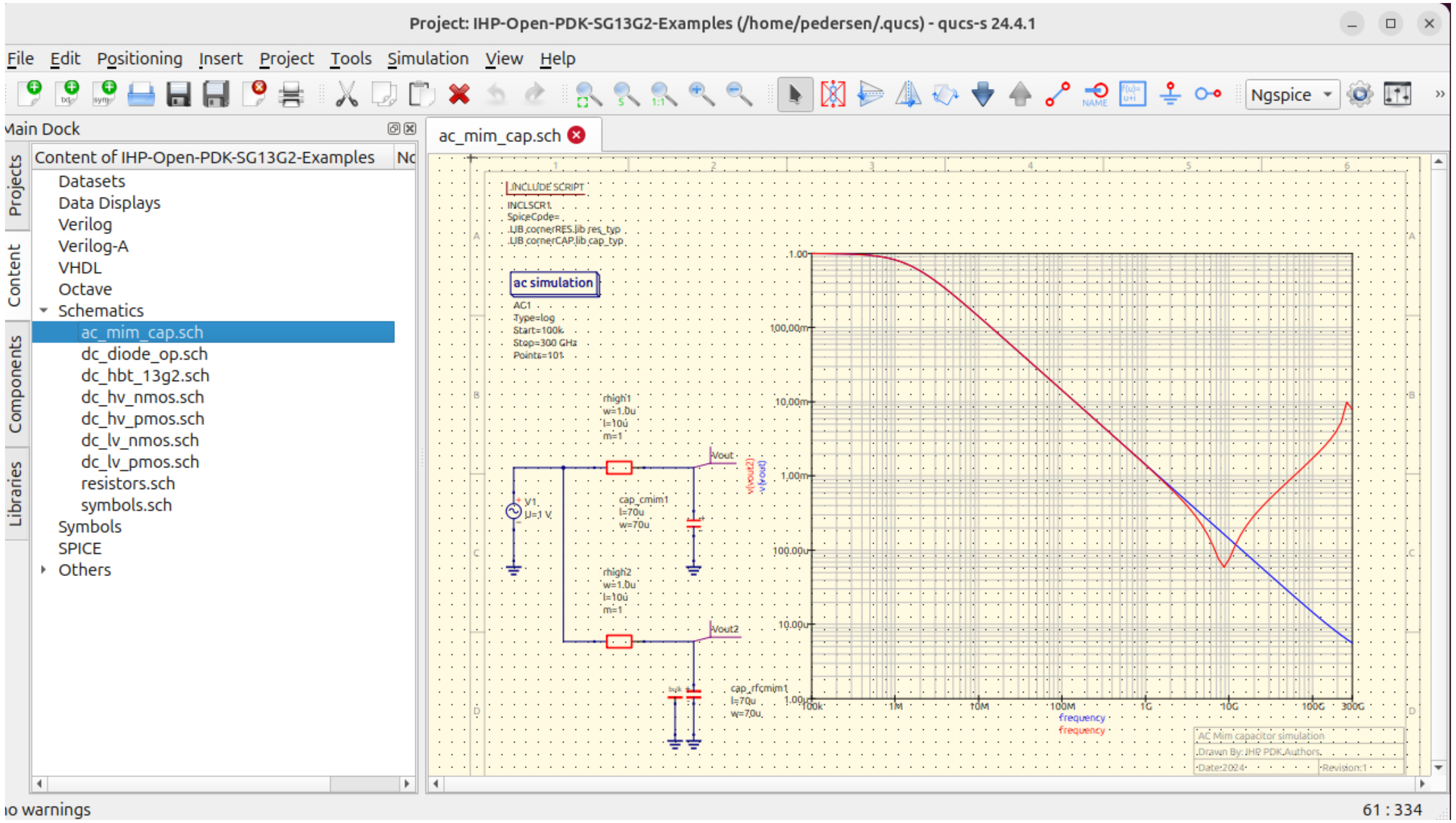


Qucs Shortcuts: https://qucs-help.readthedocs.io/en/0.0.18/short.html

Simple Navigation in QUCS-S

Running An Example

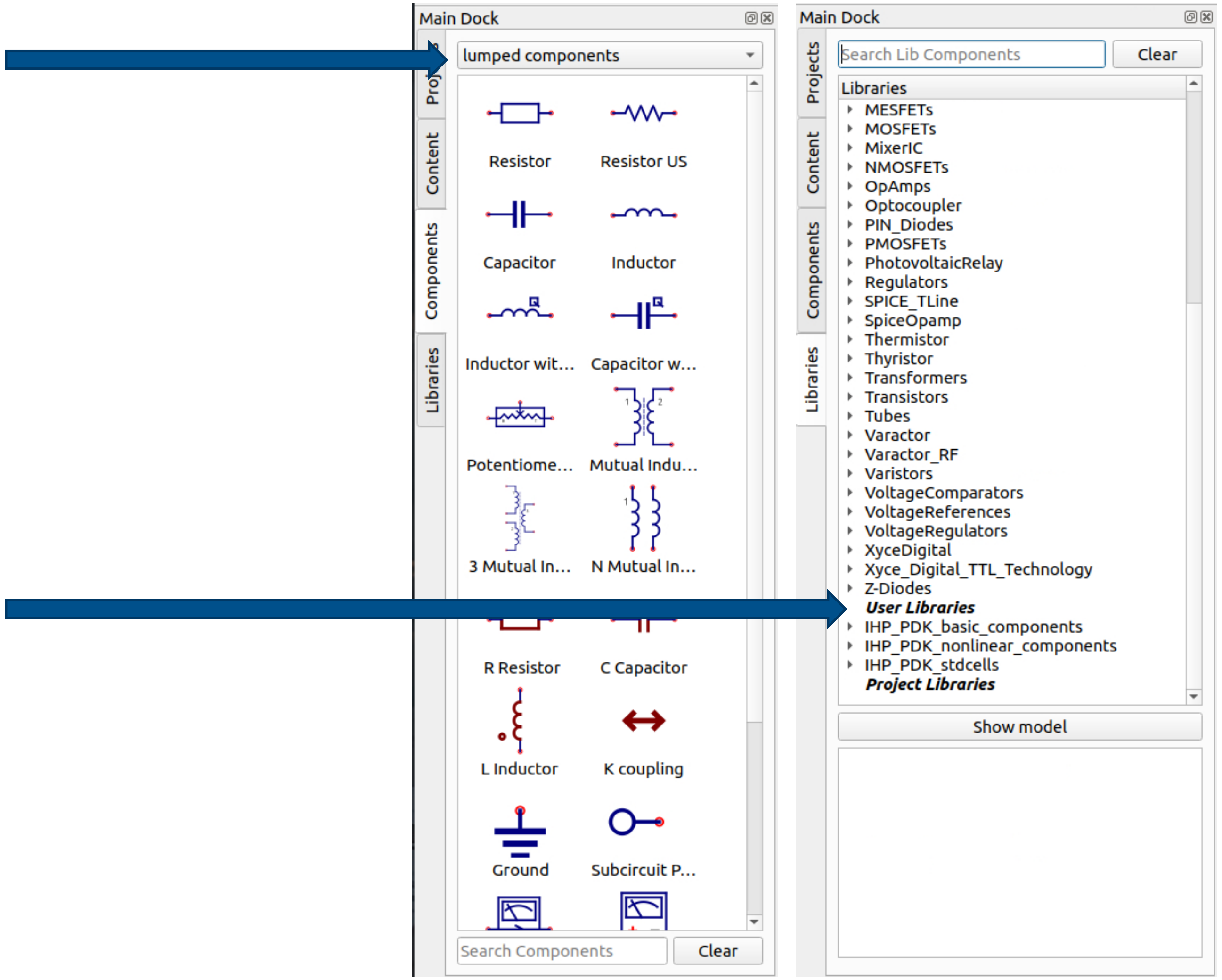




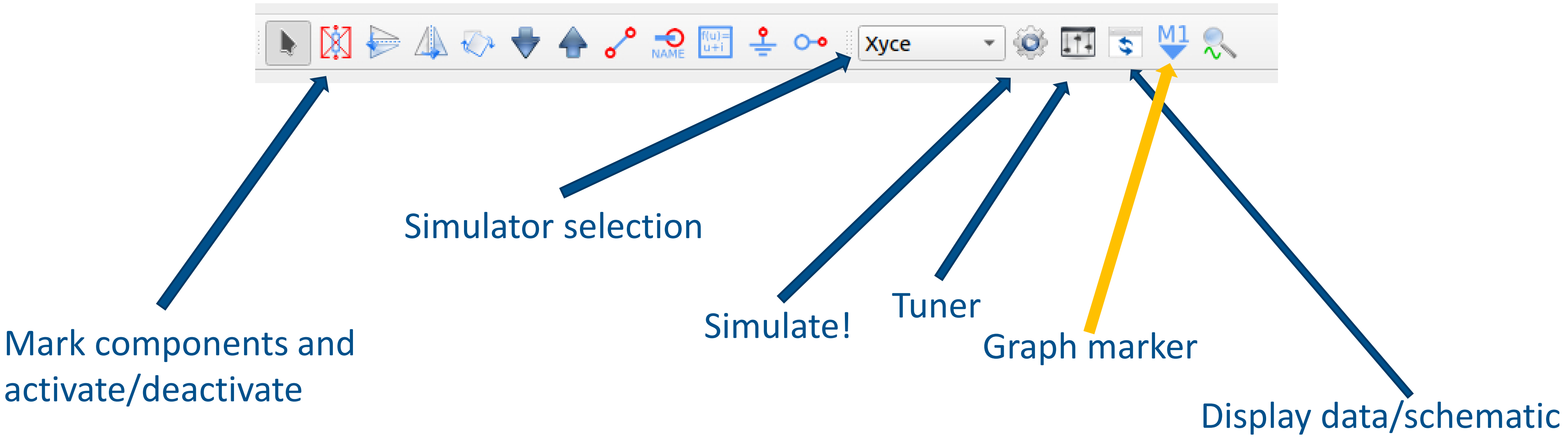
Simple Navigation in QUCS-S

Main Dock

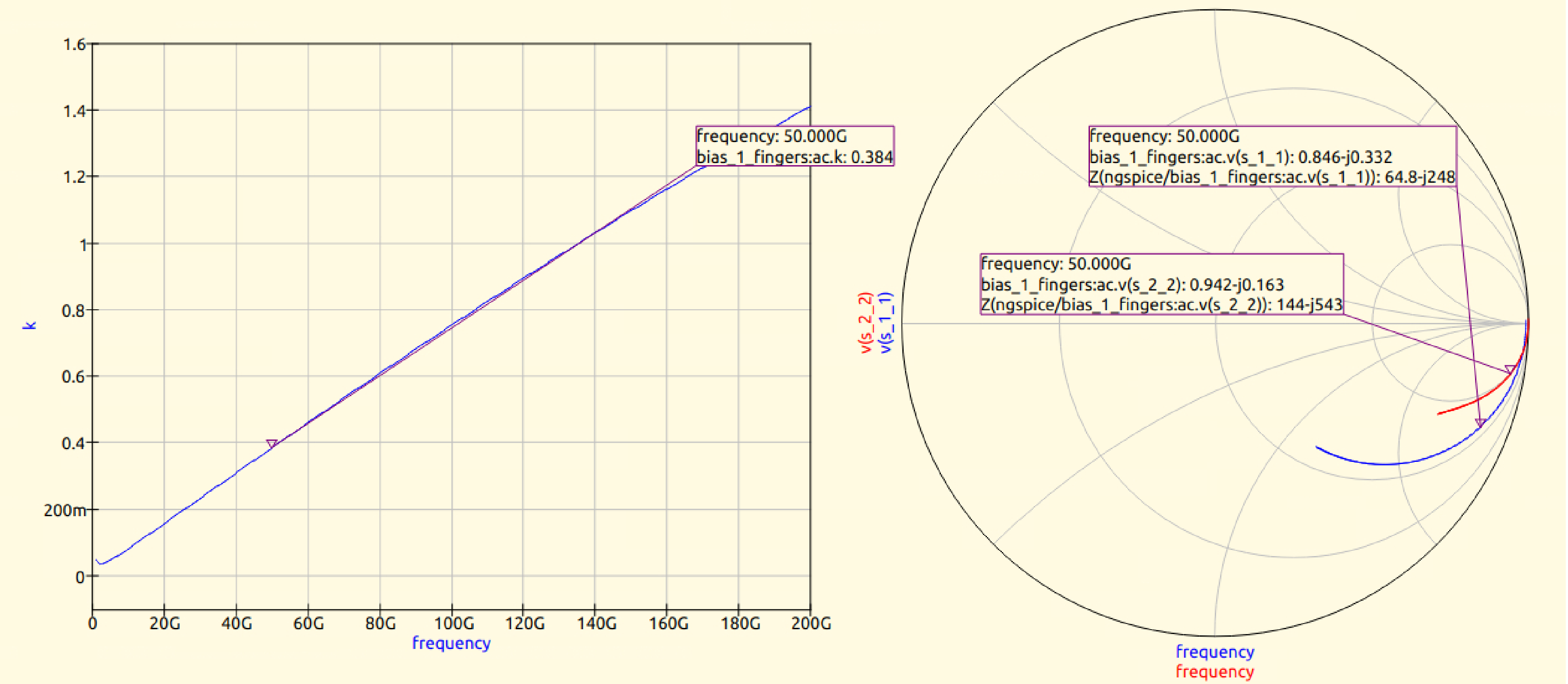
- Components: Stores standard libs and simulation blocks etc..
- The library panes stores the Open PDK components with the symbol passives (simple drag and drop)



Simple Navigation in QUCS-S

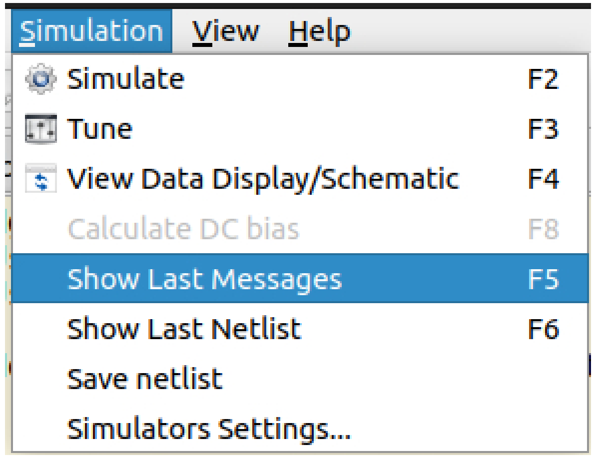


Simple Navigation in QUCS-S

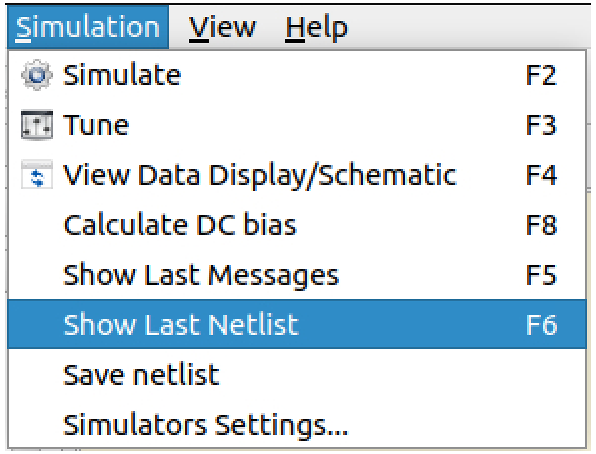


number	bias_1_fingers:v(vb)	bias_1_fingers:v(vcc)	bias_1_fingers:i(pr1)
1	1	1.65	0.0034

Simple Navigation in QUCS-S




```
Ngspice started...
Using SPARSE 1.3 as Direct Linear Solver
Using SPARSE 1.3 as Direct Linear Solver

Note: No compatibility mode selected!


Circuit: * qucs 24.4.1  /home/pedersen/projects/ihp-analogacademy/modules/module_2_50ghz_mpa/part_1_biasing/schematic/bias_1_fingers/bias_1_fingers.sch

Doing analysis at TEMP = 27.000000 and TNOM = 27.000000


No. of Data Rows : 200
binary raw file "spice4qucs.sp1.plot"
Reset re-loads circuit * qucs 24.4.1  /home/pedersen/projects/ihp-analogacademy/modules/module_2_50ghz_mpa/part_1_biasing/schematic/bias_1_fingers/bias_1_fingers.sch

Circuit: * qucs 24.4.1  /home/pedersen/projects/ihp-analogacademy/modules/module_2_50ghz_mpa/part_1_biasing/schematic/bias_1_fingers/bias_1_fingers.sch

Doing analysis at TEMP = 27.000000 and TNOM = 27.000000


No. of Data Rows : 1
Reset re-loads circuit * qucs 24.4.1  /home/pedersen/projects/ihp-analogacademy/modules/module_2_50ghz_mpa/part_1_biasing/schematic/bias_1_fingers/bias_1_fingers.sch

Circuit: * qucs 24.4.1  /home/pedersen/projects/ihp-analogacademy/modules/module_2_50ghz_mpa/part_1_biasing/schematic/bias_1_fingers/bias_1_fingers.sch

ngspice-43 done
```




```
* Qucs 24.4.1  /home/pedersen/projects/IHP-AnalogAcademy/modules/module_2_50GHz_MPA/part_1_biasing/schematic/bias_1_fingers/bias_1_fingers.sch
.INCLUDE "/usr/local/share/qucs-s/xspice_cmlib/include/ngspice_mathfunc.inc"

.SUBCKT IHP_PDK_nonlinear_components_npn13G2  gnd c b e bn Nx=1
X1 c b e bn npn13G2 Nx={Nx}
.ENDS

.LIB cornerHBT.lib hbt_typ

VP3 _net0 0 dc 0 ac 0.632456 SIN(0 0.632456 1MEG) portnum 1 z0 50
C11 _net0 _net1 1U
L5 _net2 _net1 1U
L8 _net3 Vcc 1U
VPr1 _net3 Collector_voltage DC 0
V3 _net2 0 DC 0.97
C12 _net3 _net4 1U
VP4 _net4 0 dc 0 ac 0.632456 SIN(0 0.632456 1MEG) portnum 2 z0 50
V1 Vcc 0 DC 1.65

Xnpn13G3 0  Collector_voltage _net1 0 gnd IHP_PDK_nonlinear_components_npn13G2 Nx=10

.control

SP LIN 200 1G 200G
let k = (1 - abs(s_1_1)^2 - abs(s_2_2)^2 + abs(s_1_1 * s_2_2 - s_1_2 * s_2_1)^2) / (2 * abs(s_1_2 * s_2_1))

write spice4qucs.sp1.plot S_1_1 Y_1_1 Z_1_1 S_1_2 Y_1_2 Z_1_2 S_2_1 Y_2_1 Z_2_1 S_2_2 Y_2_2 Z_2_2 k
destroy all
reset

op
print v(Collector_voltage) i(VPr1) v(Vcc)   > spice4qucs.dc1.ngspice.dc.print
destroy all
reset

exit
.endc
.END
```

Catching Up / Lunch Time !

Next session:
Biasing

Leibniz Institute
for high
performance
microelectronics

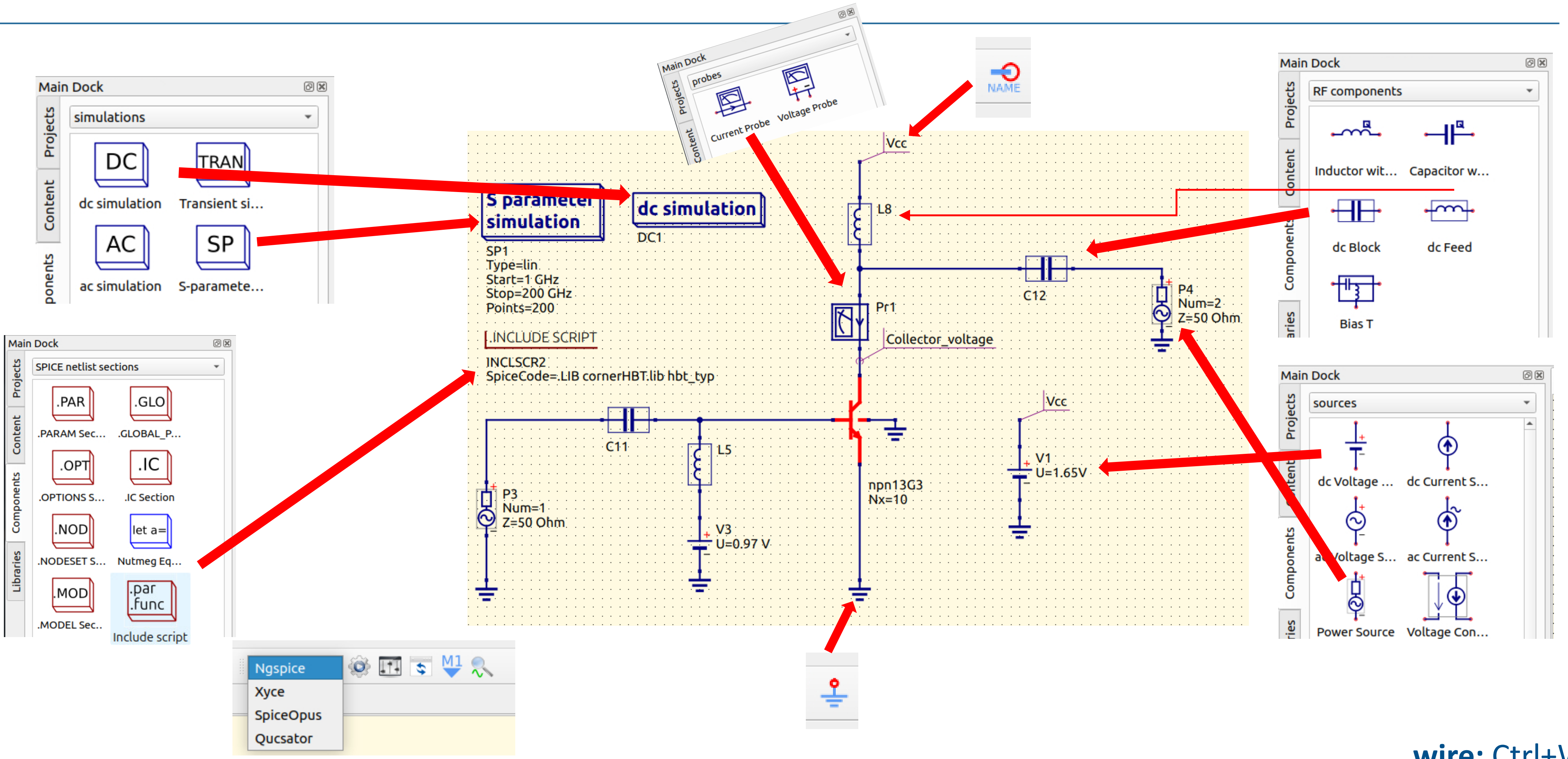
Part 1

Biasing

Reference: modules/module_2_50GHz_MPA/part_1_biasing

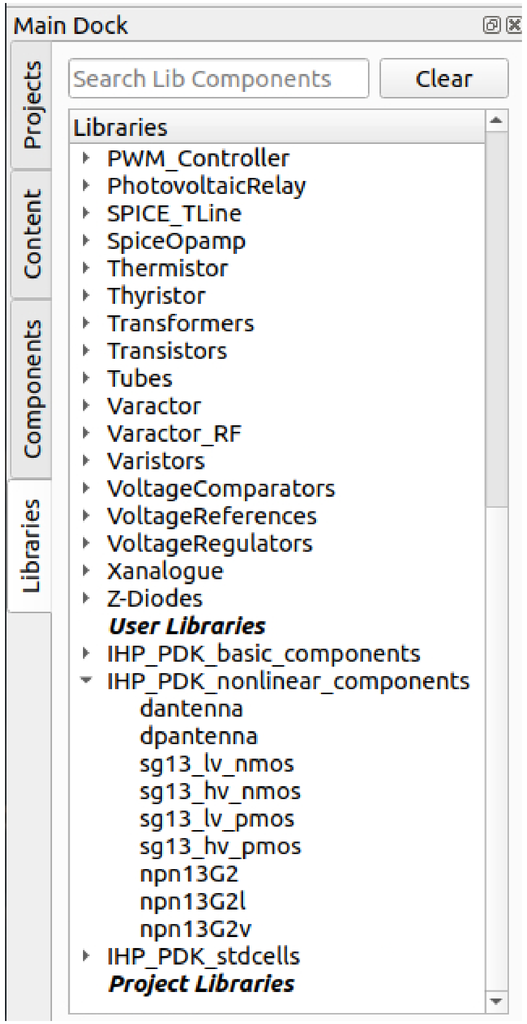
Leibniz Institute
for high
performance
microelectronics

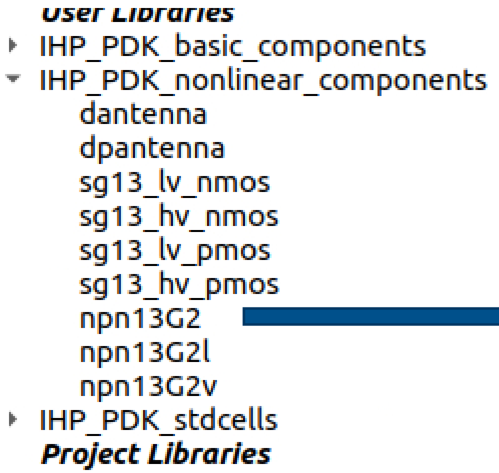
Building The Schematic

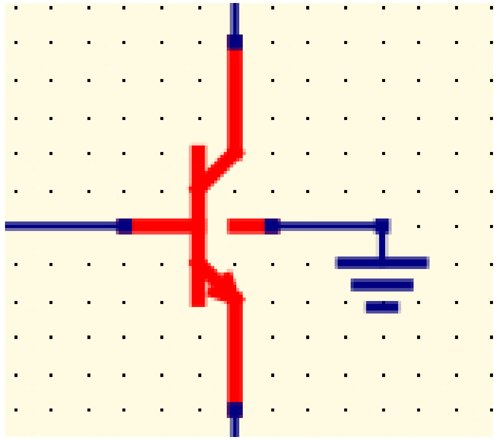


wire: Ctrl+W

Building The Schematic



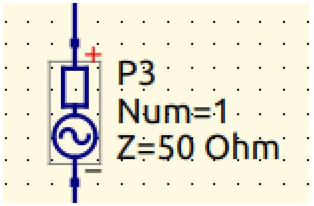




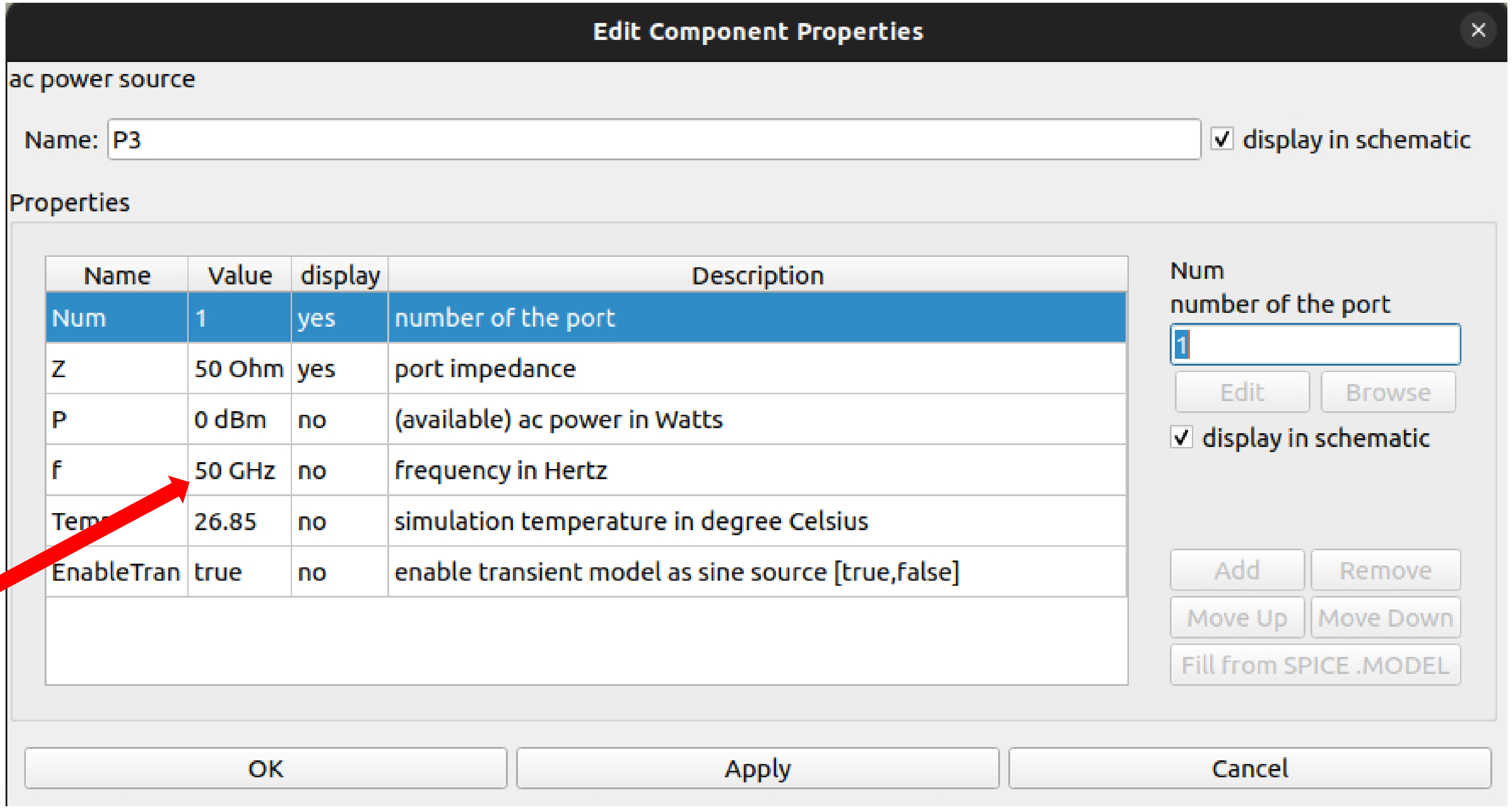
Check Component Parameters

You have to set different parameters that aren't necessarily shown!

Example:



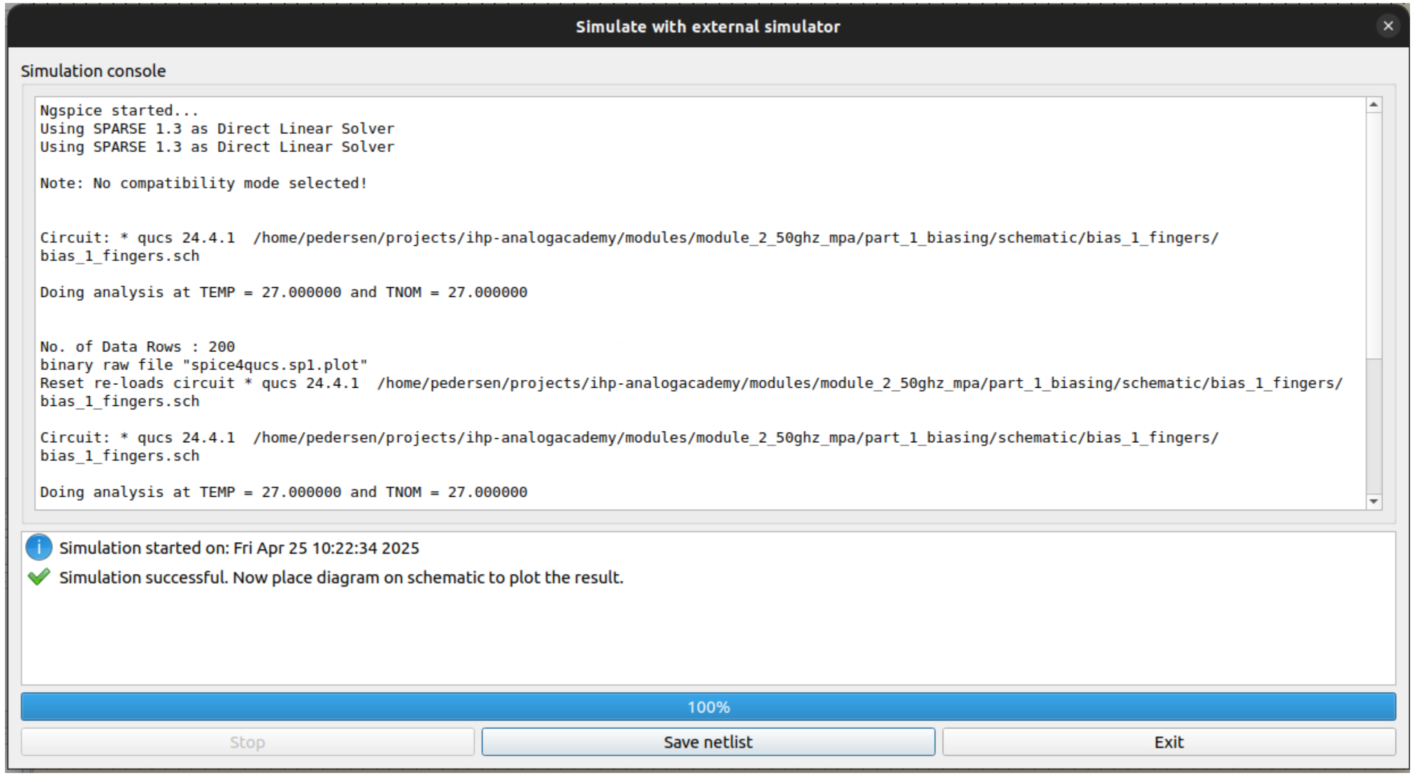
Double Click



!!!

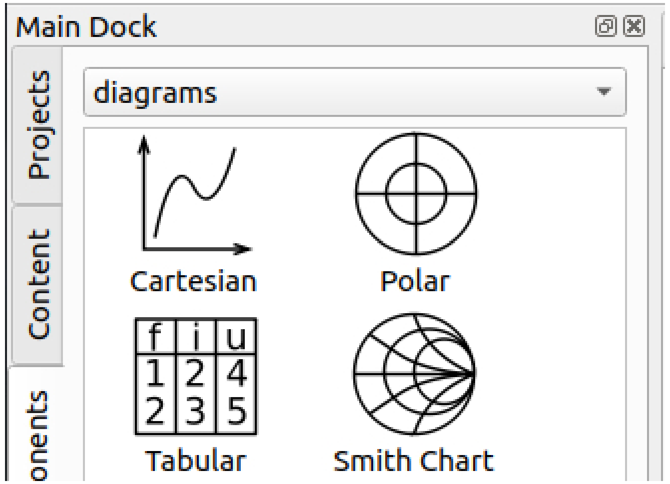
Same for Port 2

Running The Simulation

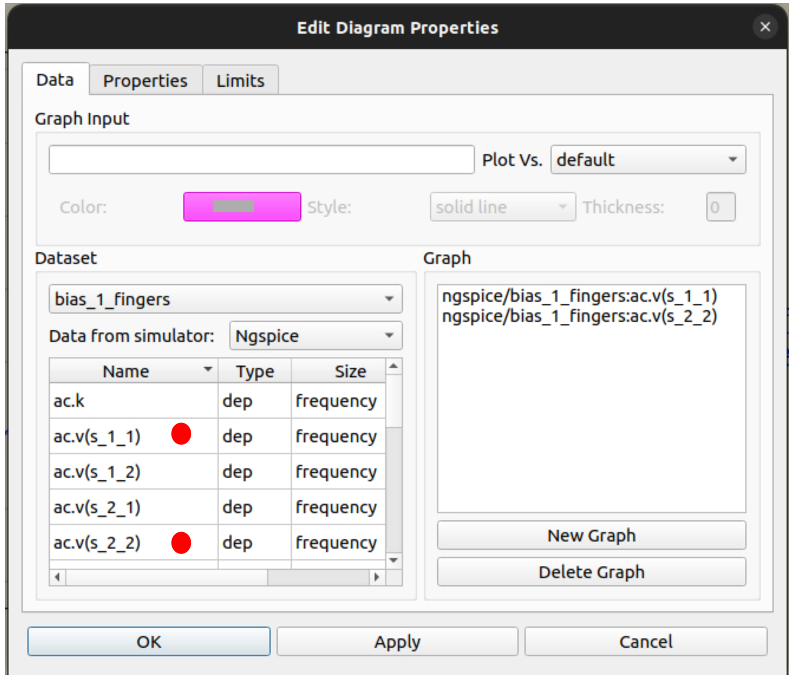


Viewing The First Results

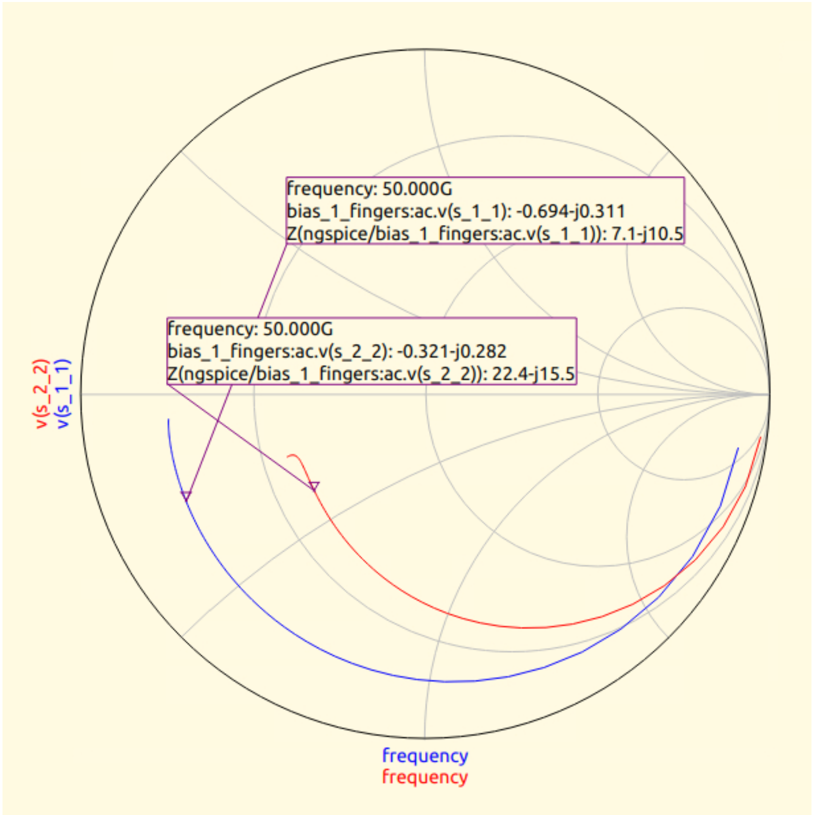
Start by switching to data display



Select the Smith Chart

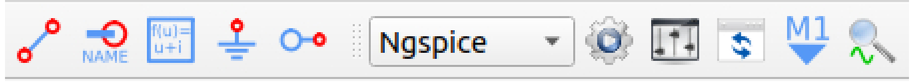


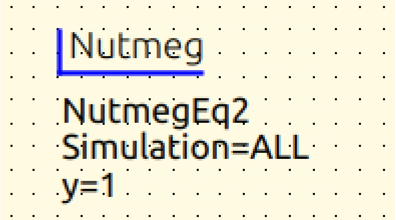
Double click the S11 and S22

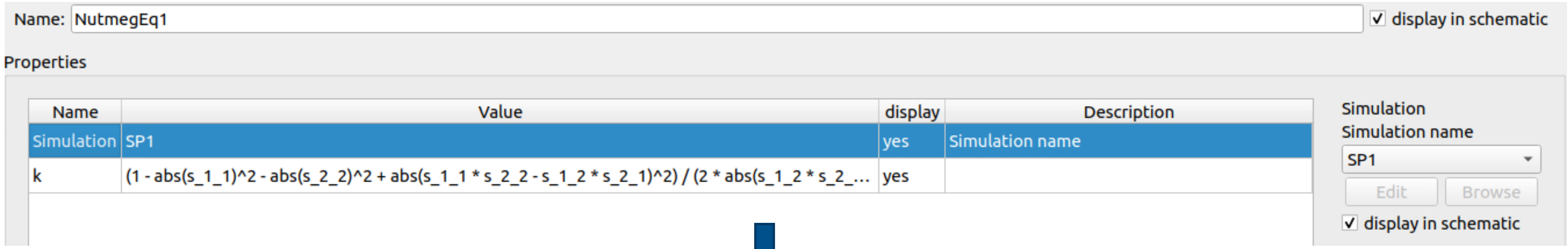


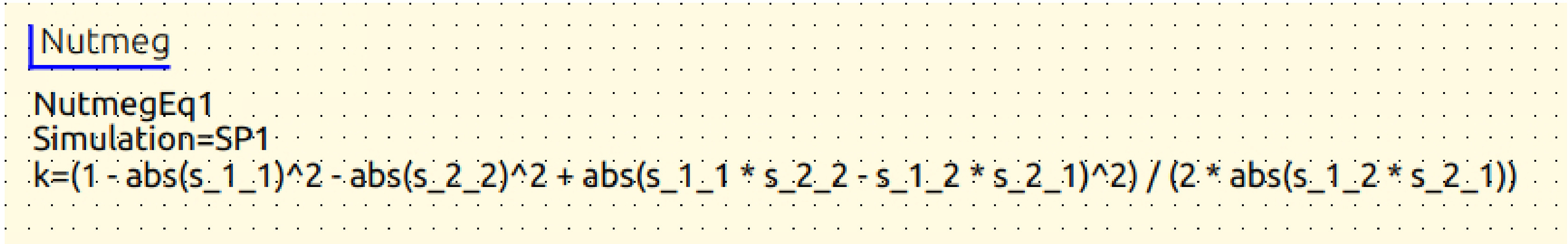
View the data and insert markers with M1
Use arrow keys to move the marker

Inserting Equation





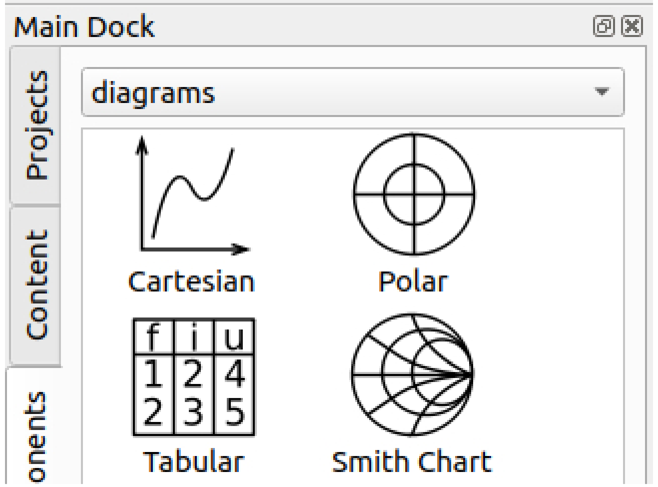




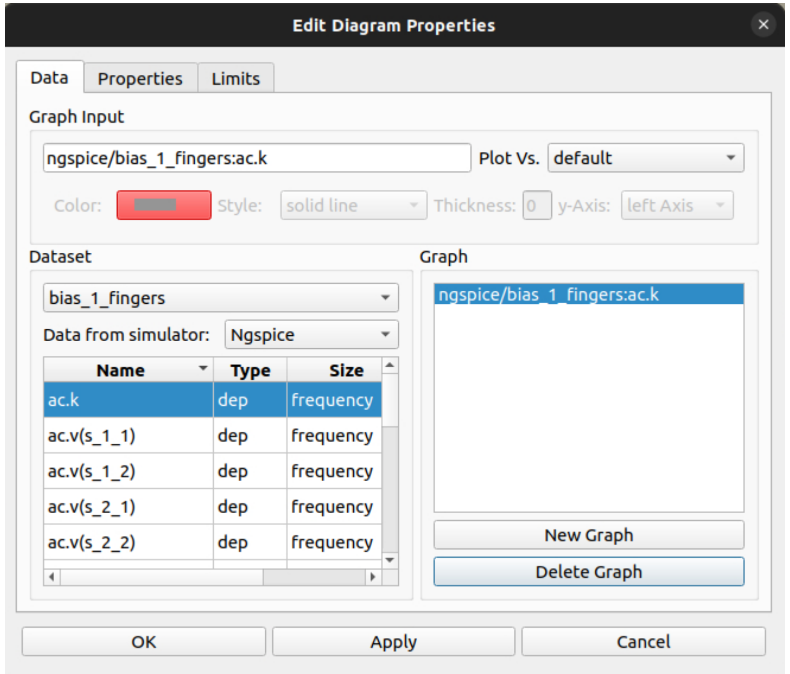
Stability / K-Factor

k=(1 - abs(s_1_1)^2 - abs(s_2_2)^2 + abs(s_1_1 * s_2_2 - s_1_2 * s_2_1)^2) / (2 * abs(s_1_2 * s_2_1))

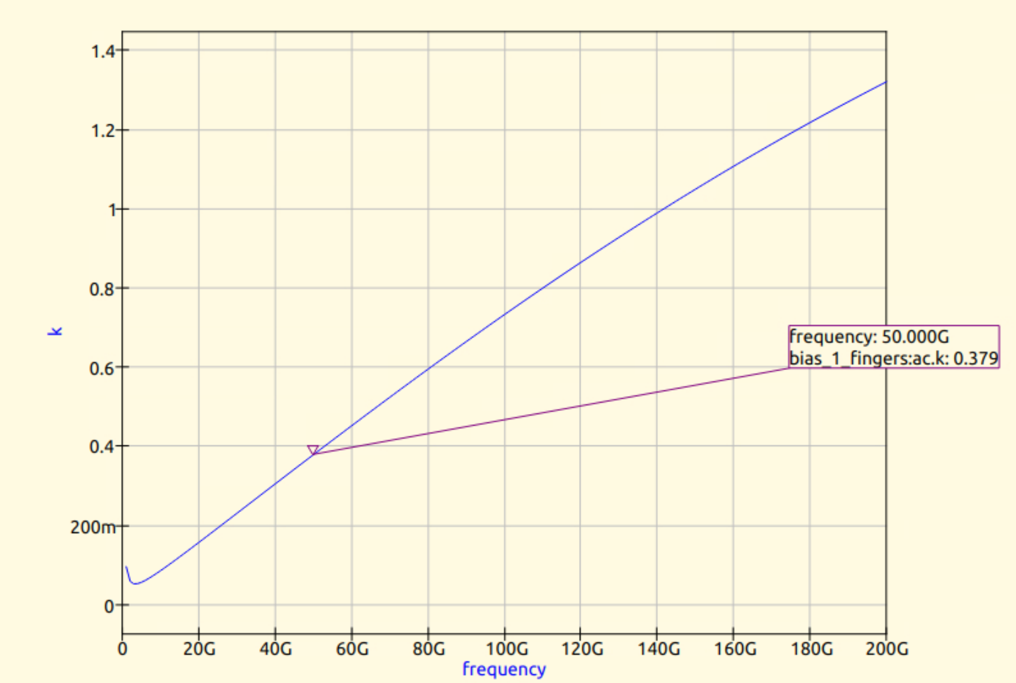
Viewing The K-Factor



Select the Cartesian plot

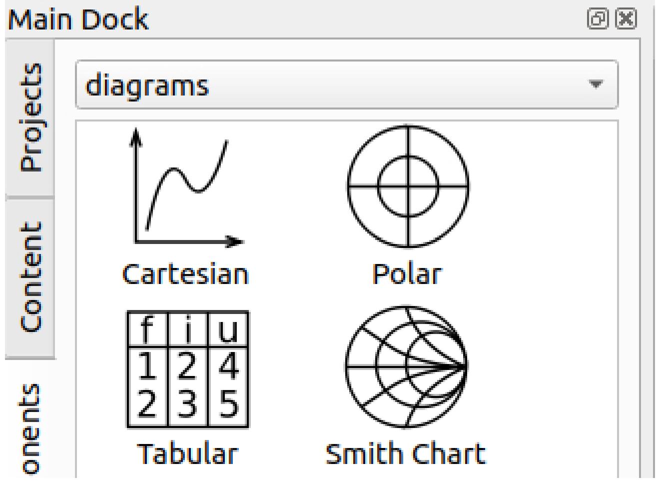


Select the K-factor

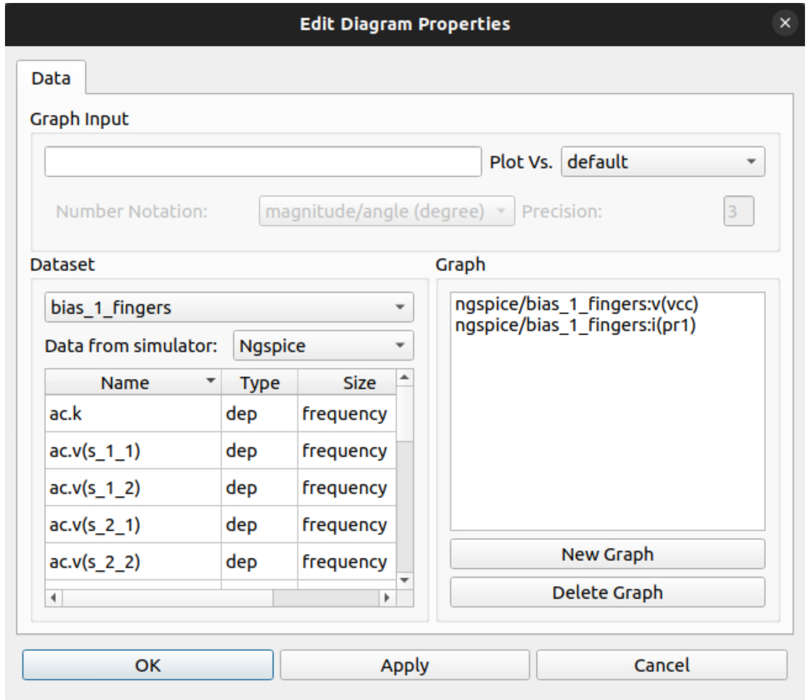


Clearly not stable!

Viewing Data From Table



Select tabular



Select the collector voltage and current

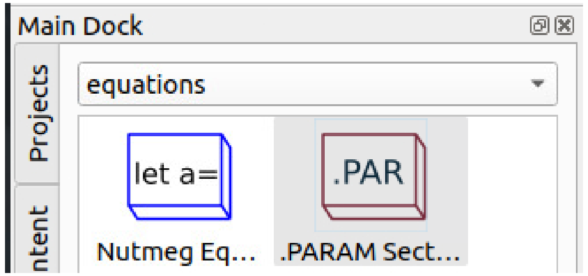
number	bias_1_fingers:v(vcc)	bias_1_fingers:i(pr1)
1	1.65	0.0269

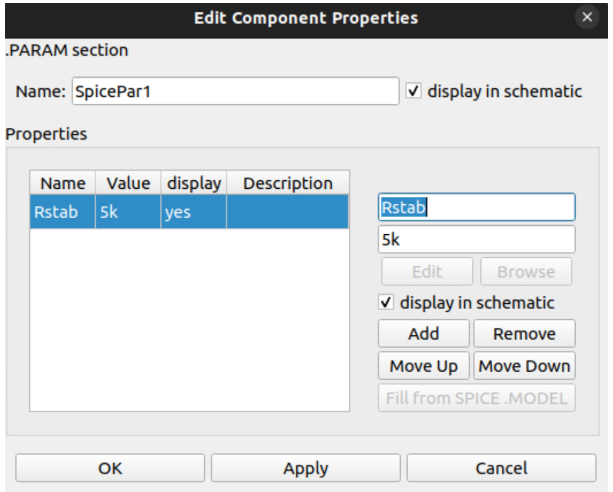
Checking some bias points

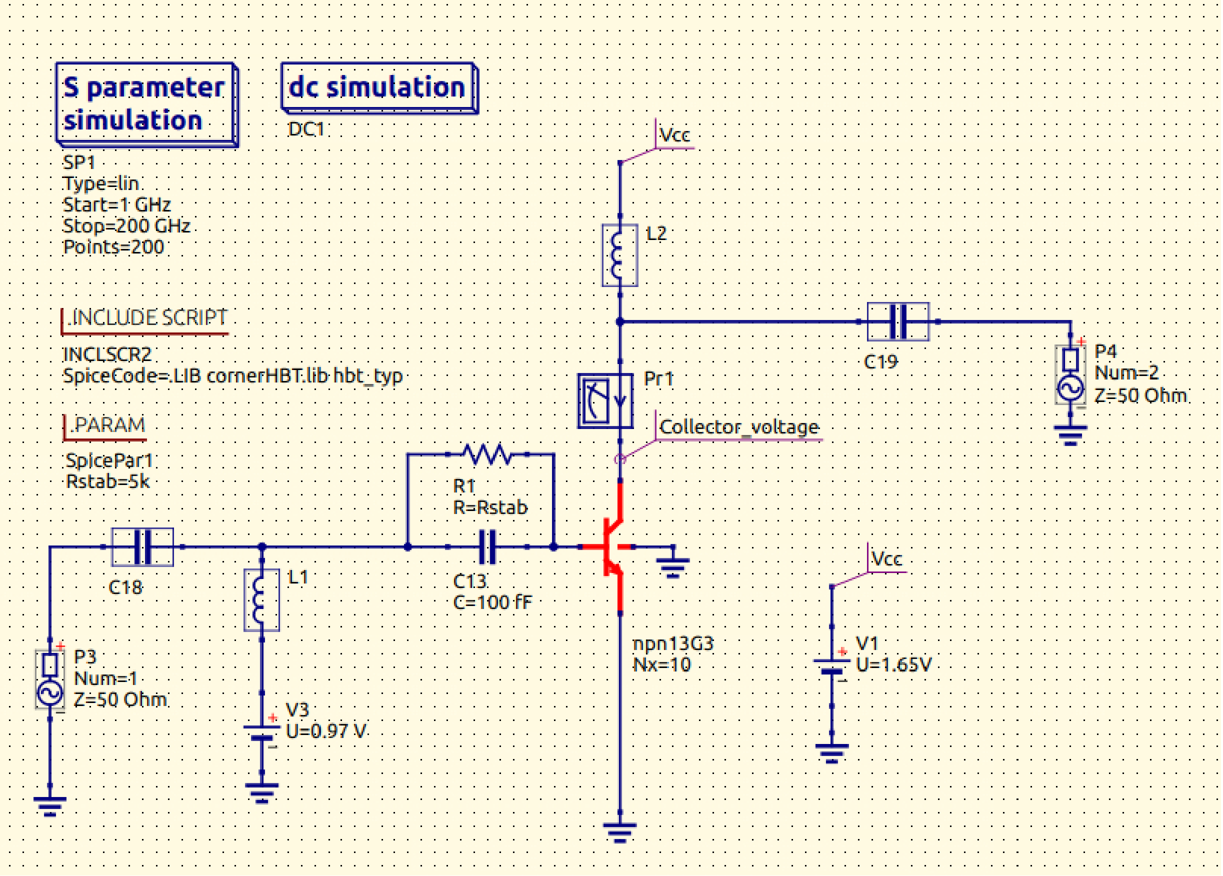
Tuning Components

Introducing Stability Comps

- We want to tune the resistance of the base resistor!
- We introduce a .Param block:



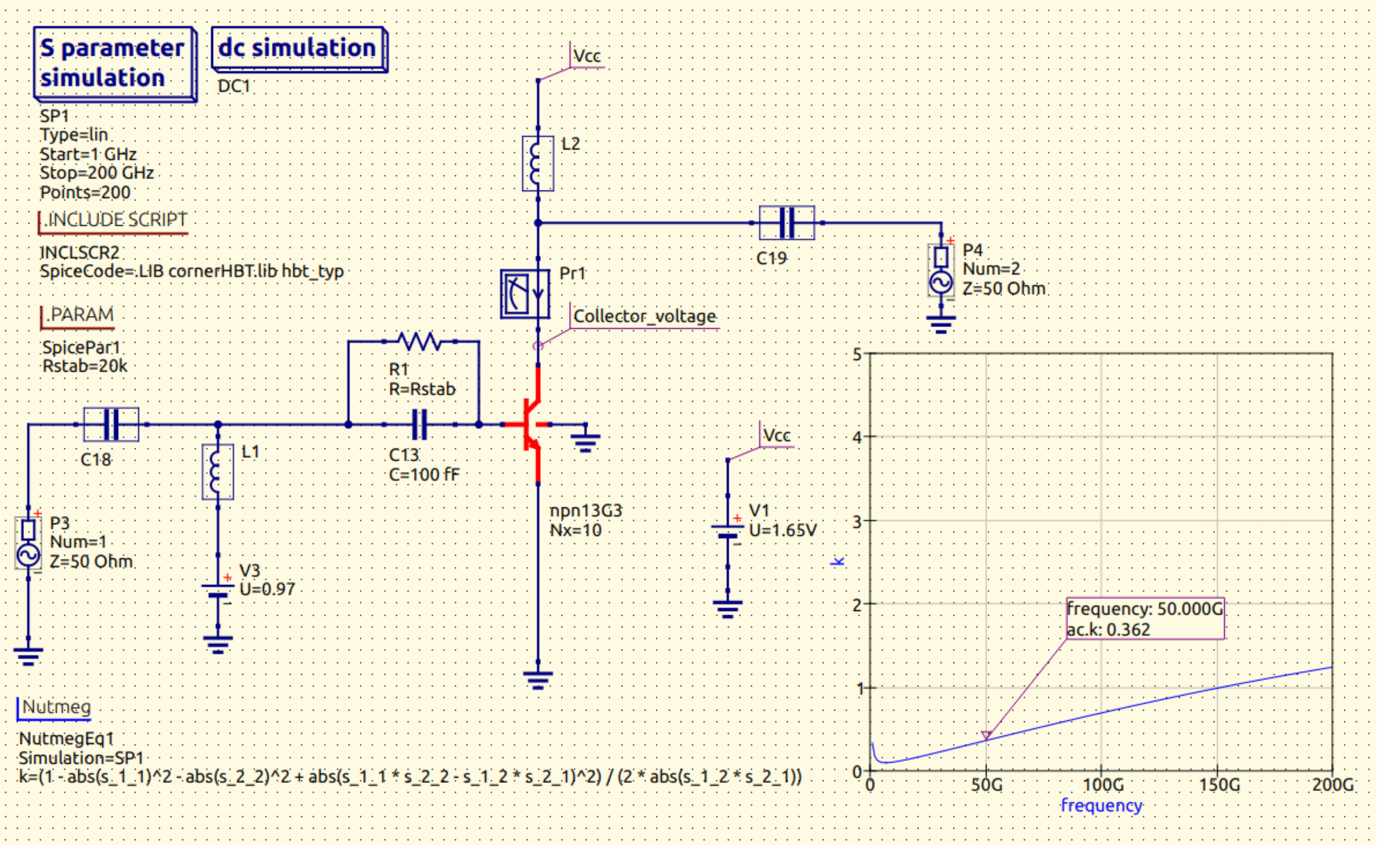


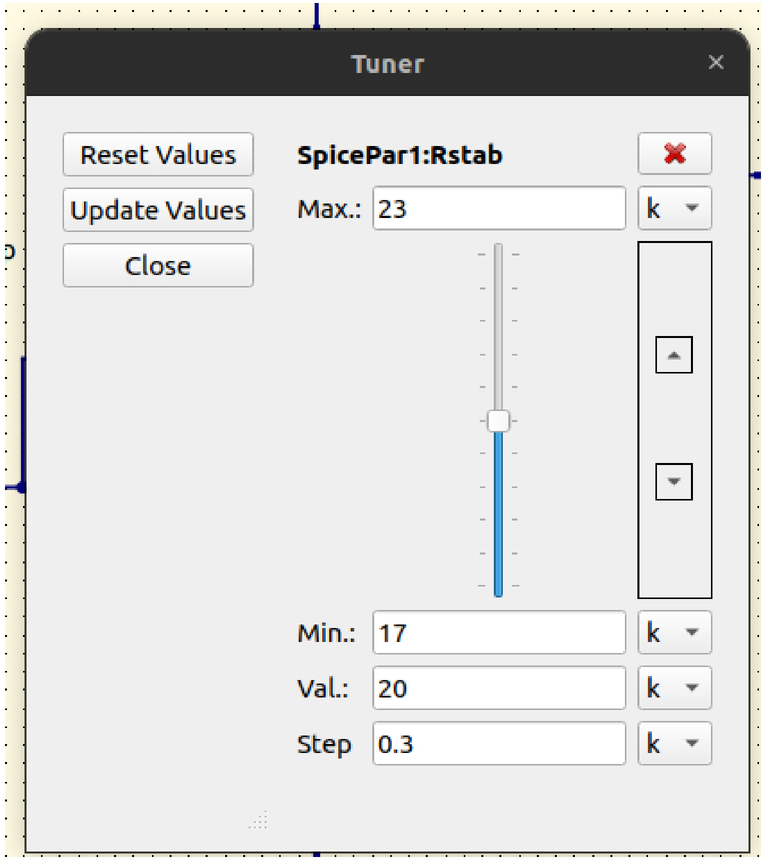


- Also set the parameter in the resistor

Tuning Components

Instantiate the Cartesian plot inside the schematic and view the K-factor From here open the tuning function





Select the Rstab parameter in the .param block and start tuning

Stabilize The Amplifier ☺

- Try as best as you can to stabilize the amplifier
- You can employ any method you like (emitter degeneration, biasing etc…)
- Its not critical to stabilize the amplifier since we only want to create a appropriate flow ☺
- If unsure, stabilize the amp by adjusting components for **K > 1** at 50 GHz.

Leibniz Institute
for high
performance
microelectronics

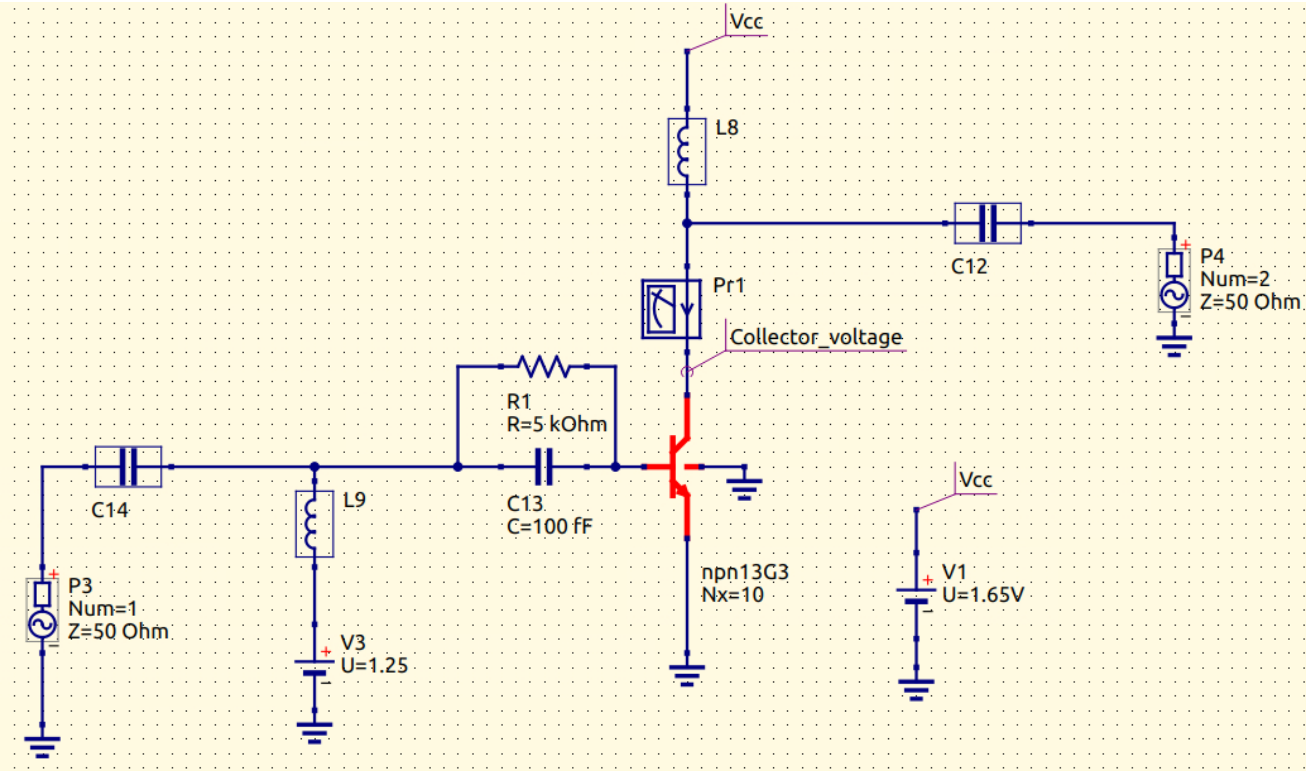
Part 2

Matching

Reference: modules/module_2_50GHz_MPA/part_2_matching_ideal

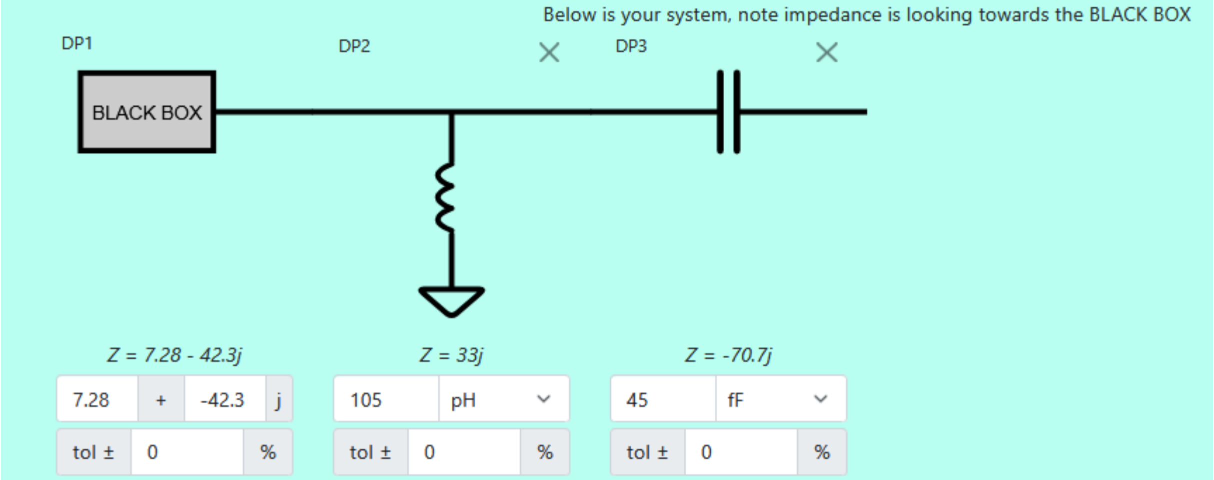
Matching

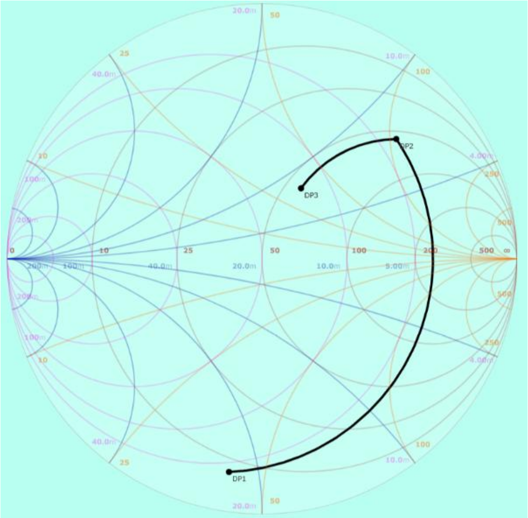
- To move forward we will use the schematic seen to the right
- This is simply done to keep quite consistent with the first iteration of the design earlier
- You may continue with your own schematic to keep better stability factor ☺



Online Matching Tool

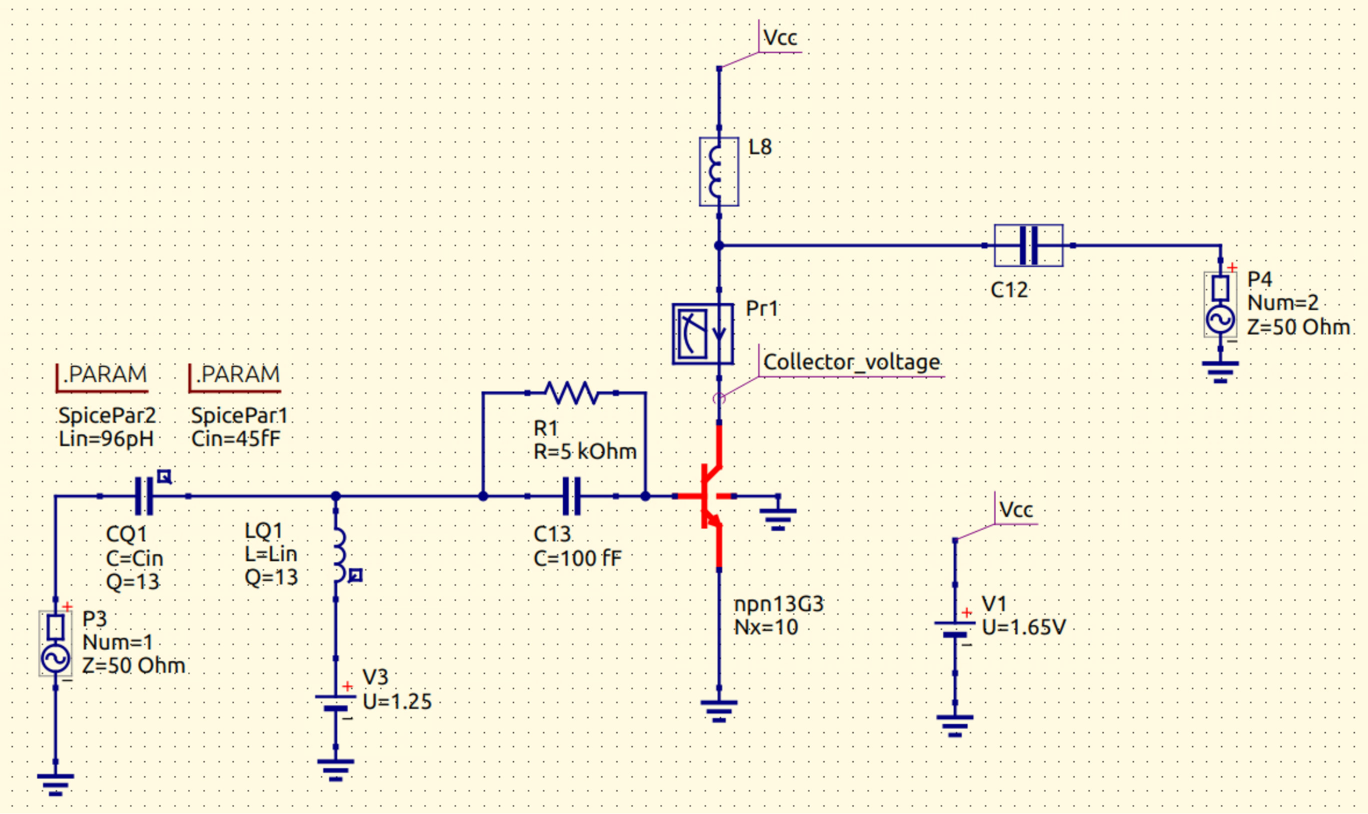
- At this point you can navigate to the following link to match your input section: *https://www.will-kelsey.com/smith_chart/*
- You can do the matching in QUCS-S using the matching tool under tools, but not for this kind of section that we wish to create
- When an appropriate matching have been found you can refer to the next slide to insert the components and tune the parameters



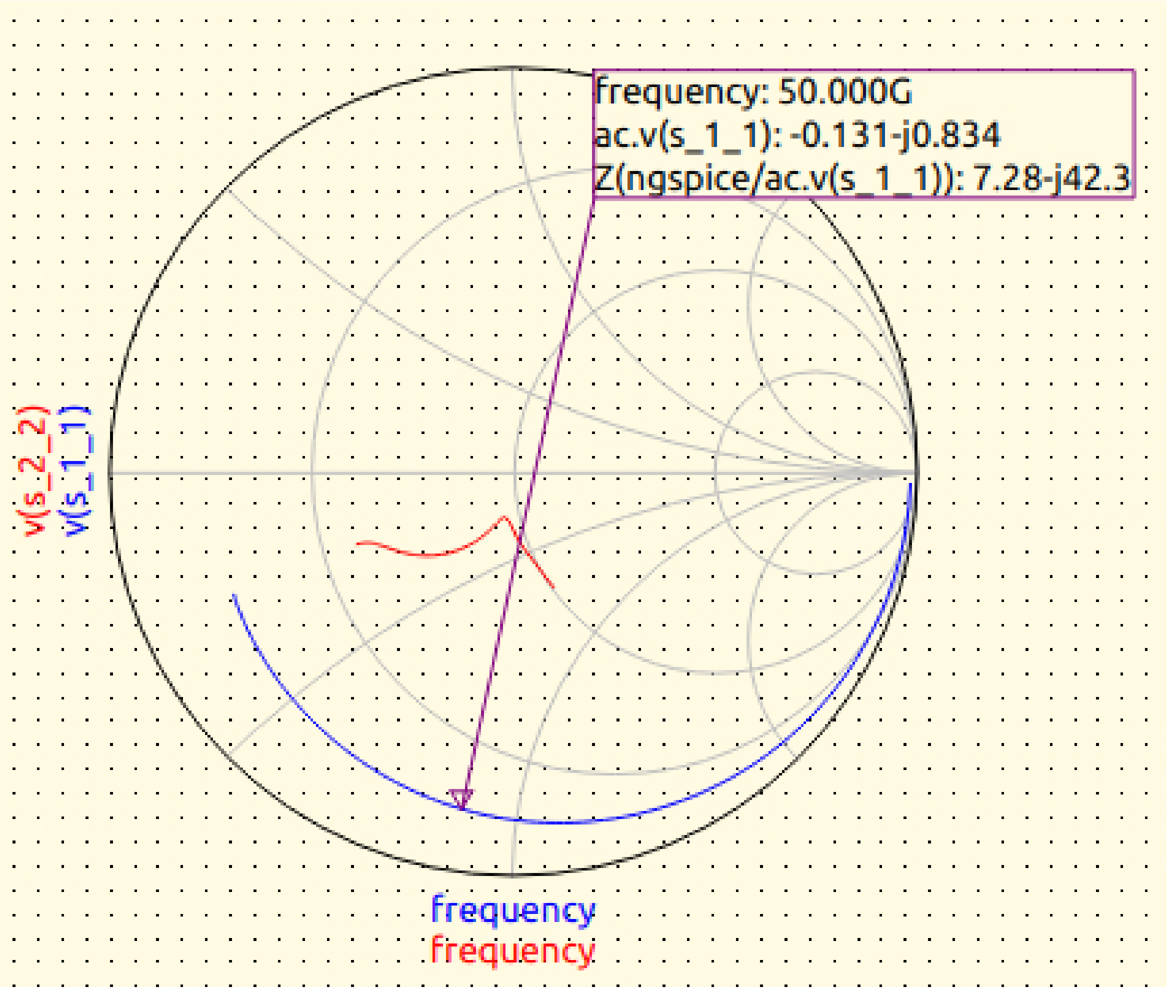


Matching

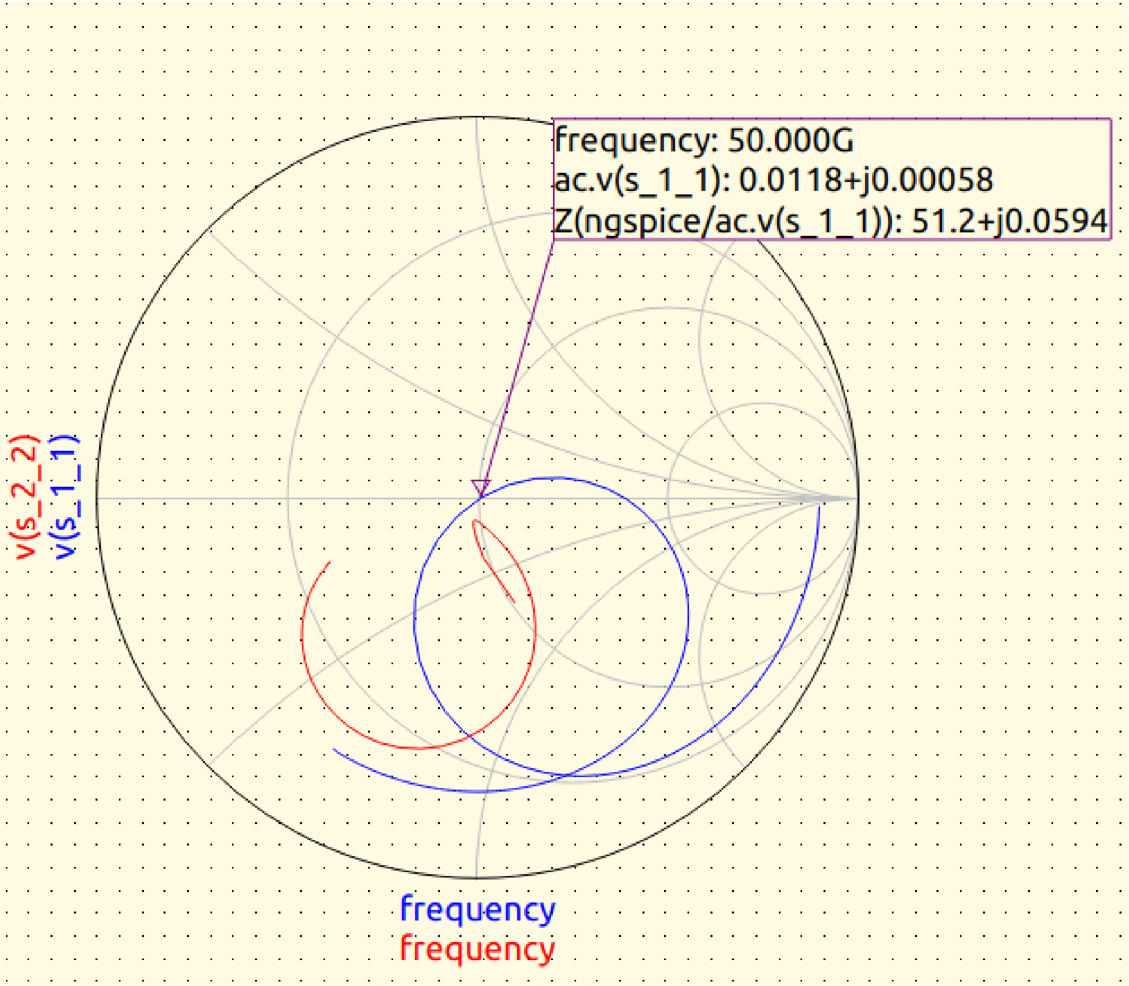
- We will start by inserting “real” capacitor and inductor in the input and (remember to set frequency of Q-factor components)
- Since we define variables for the capacitance and inductance we also need to include **.param** blocks for this
- This will be used to tune the input sizes to get best input matching
- **C** and **L** values are tuned based on parameters from the previous slide—yours may vary ☺



Matching



frequency: 50.000G
ac.v(s_1_1): -0.131-j0.834
Z(ngspice/ac.v(s_1_1)): 7.28-j42.3
v(s_2_2)
v(s_1_1)
frequency
frequency



frequency: 50.000G
ac.v(s_1_1): 0.0118+j0.00058
Z(ngspice/ac.v(s_1_1)): 51.2+j0.0594
v(s_2_2)
v(s_1_1)
frequency
frequency

Part 3

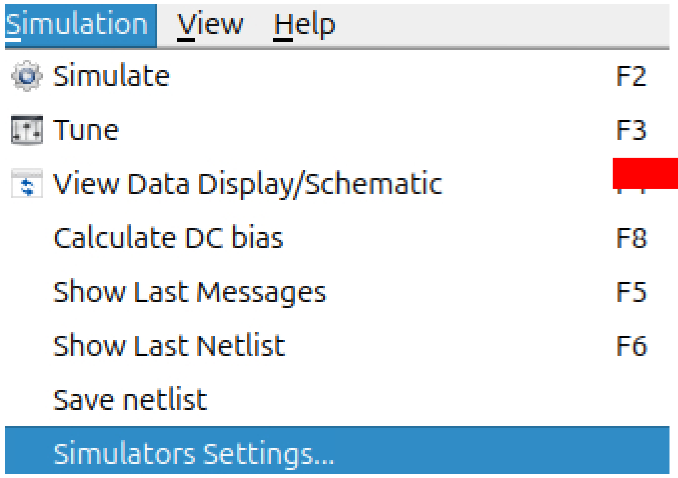
Matching

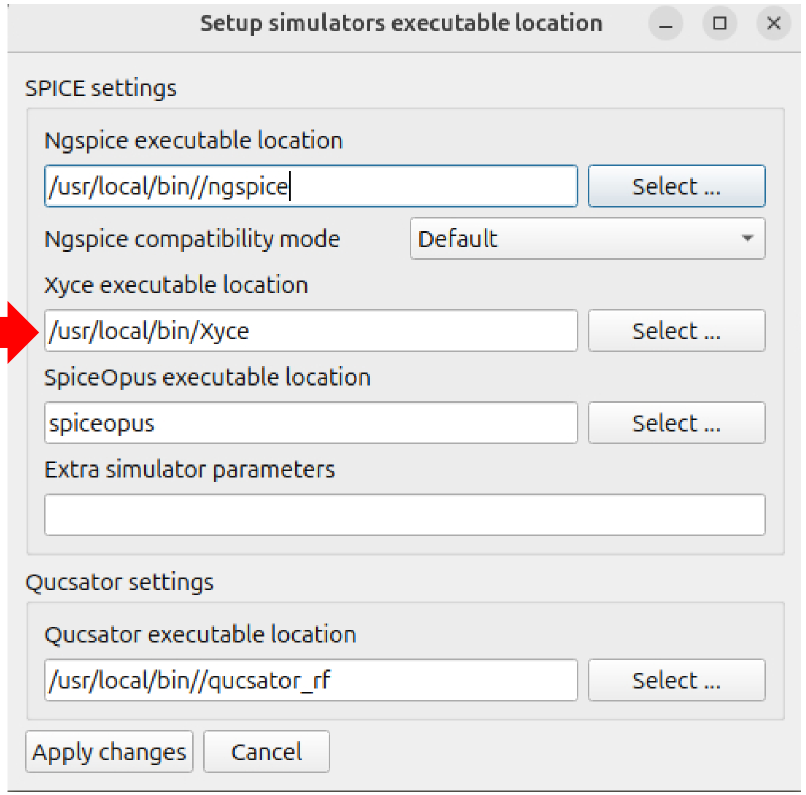
Reference: modules/module_2_50GHz_MPA/part_3_nonlinear_analysis

Leibniz Institute for high performance microelectronics

Xyce Setup

- Make sure the path to the simulator is set correctly!



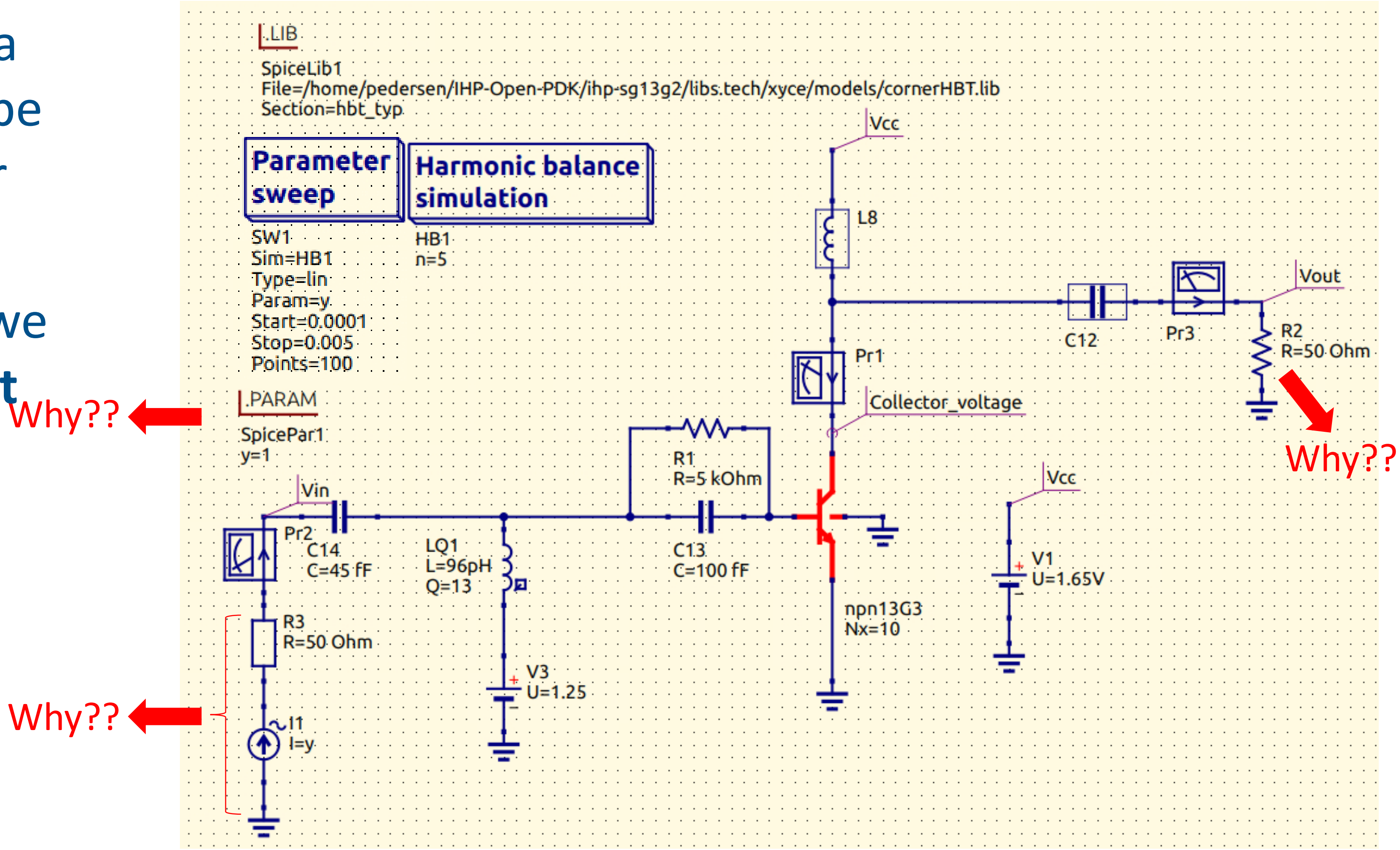


Nonlinear Analysis

- We switch simulator to Xyce
- This imposes some issues!!
- Math calculations for plotting is not possible!
- Sweeping is somehow challenging
- Pushing to hard into the Nonlinear Domain corrupts the data from xyce
- Lets look at the schematic setup on the next slide!

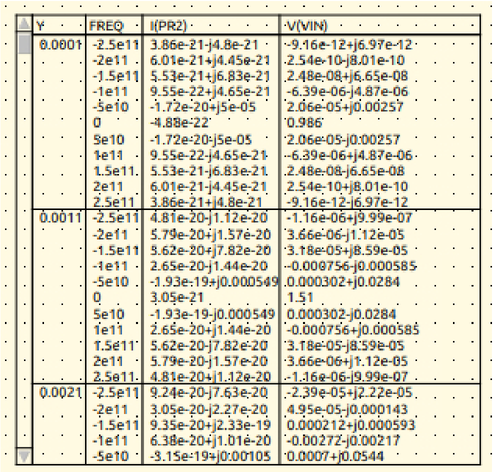
Nonlinear Analysis (Compression Point)

- We node we have to include a parameter sweep which can be found in the main dock under **simulation**
- Also for the model inclusion we have to go under **SPICE netlist section** in the main dock and choose the **.Lib directive**
- Remember to switch the simulator to Xyce!
- Run the Simulation

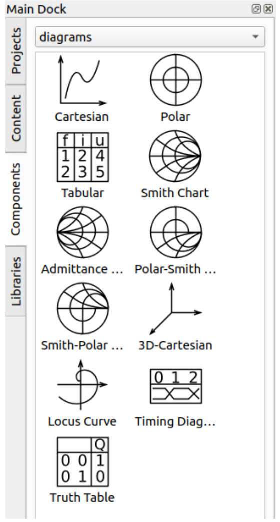


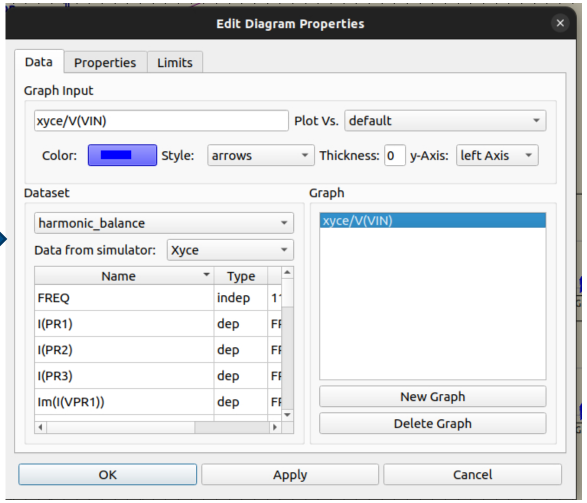
Visualizing The Results

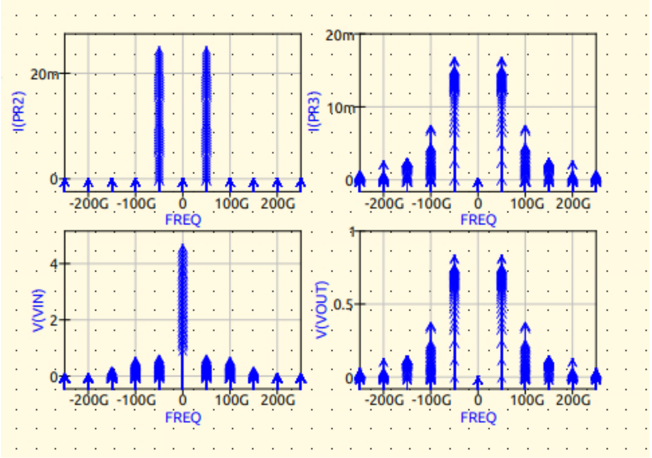
- After setting up the simulation, we can run it and check the results. Upon completion, the first task is to **instantiate a table** to inspect the output data and ensure proper formatting.
- To extract the data for post-processing we need to plot data, which we will do with a **Cartesian** plot seen on the following slide



Y	FREQ	I(PR2)	V(VIN)
0.0001	-2.5e11	3.86e-21-j4.8e-21	-9.16e-12+j6.97e-12
	-2e11	6.01e-21+j4.45e-21	2.54e-10-j8.01e-10
	-1.5e11	5.53e-21+j6.83e-21	2.48e-08+j6.65e-08
	-1e11	9.55e-22+j4.65e-21	-6.39e-06-j4.87e-06
	-5e10	-1.72e-20+j5e-05	2.06e-05+j0.00257
	0	-4.88e-22	0.986
	5e10	-1.72e-20-j5e-05	2.06e-05-j0.00257
	1e11	9.55e-22-j4.65e-21	-6.39e-06+j4.87e-06
	1.5e11	5.53e-21-j6.83e-21	2.48e-08-j6.65e-08
	2e11	6.01e-21-j4.45e-21	2.54e-10+j8.01e-10
	2.5e11	3.86e-21+j4.8e-21	-9.16e-12-j6.97e-12
0.0011	-2.5e11	4.81e-20-j1.12e-20	-1.16e-06+j9.99e-07
	-2e11	5.79e-20+j1.57e-20	3.66e-06-j1.12e-05
	-1.5e11	5.62e-20+j7.82e-20	3.18e-05+j8.59e-05
	-1e11	2.65e-20-j1.44e-20	-0.000756-j0.000585
	-5e10	-1.93e-19+j0.000549	0.000302+j0.0284
	0	3.05e-21	1.51
	5e10	-1.93e-19-j0.000549	0.000302-j0.0284
	1e11	2.65e-20+j1.44e-20	-0.000756+j0.000585
	1.5e11	5.62e-20-j7.82e-20	3.18e-05-j8.59e-05
	2e11	5.79e-20-j1.57e-20	3.66e-06+j1.12e-05
	2.5e11	4.81e-20+j1.12e-20	-1.16e-06-j9.99e-07
0.0021	-2.5e11	9.24e-20-j7.63e-20	-2.39e-05+j2.22e-05
	-2e11	3.05e-20-j2.27e-20	4.95e-05-j0.000143
	-1.5e11	9.35e-20+j2.33e-19	0.000212+j0.000593
	-1e11	6.38e-20+j1.01e-20	-0.00272-j0.00217
	-5e10	-3.15e-19+j0.00105	0.0007+j0.0544



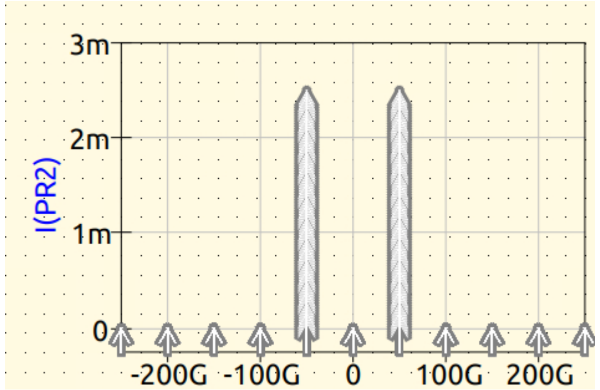


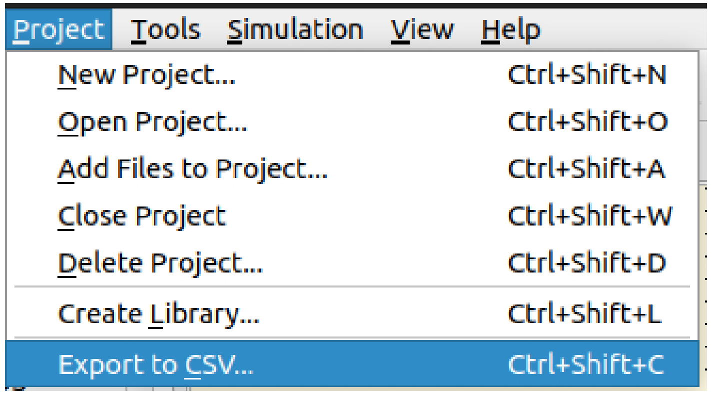


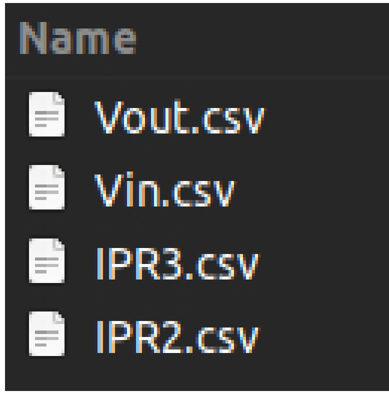
Visualizing The Results

- Mark the data on the graphs and navigate to the following
- Once the all the data for the parameters have been saved, the Jupyter-lab file at the following path can be opened: *module_2_50GHz_MPA/part_3_nonlinear_analysis/python*

  jupyter lab post_processing.ipynb







Visualizing The Results

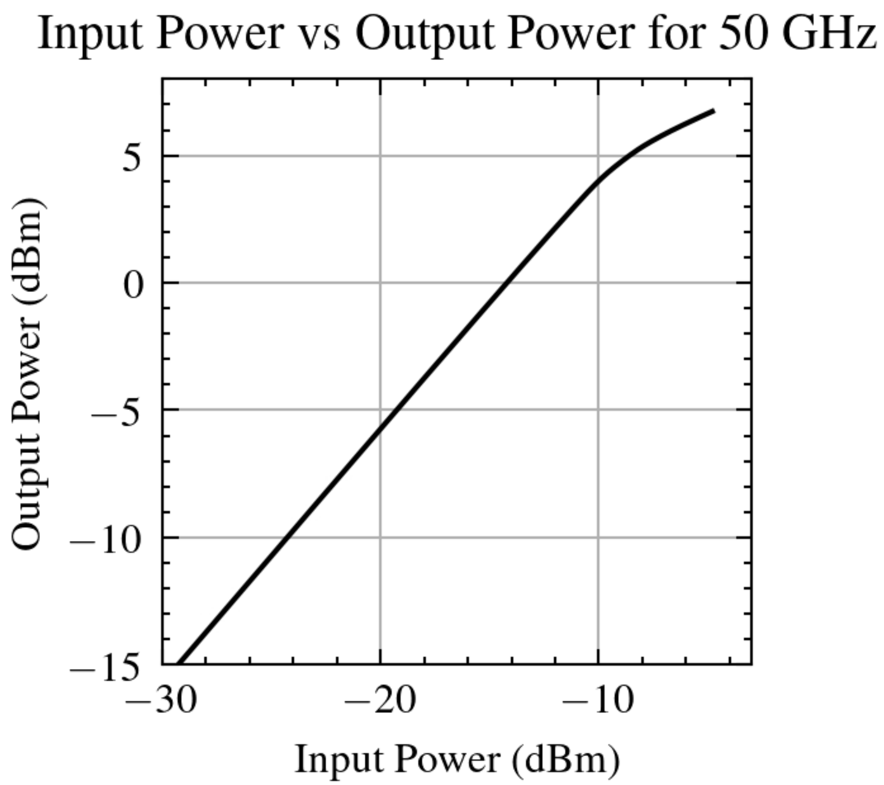
- Demonstrates how to plot extracted data using Python
- Ensure parameter names match those specified for successful extraction
- If schematic labels differ from the slides, open the CSV files to check and adjust the parameter names

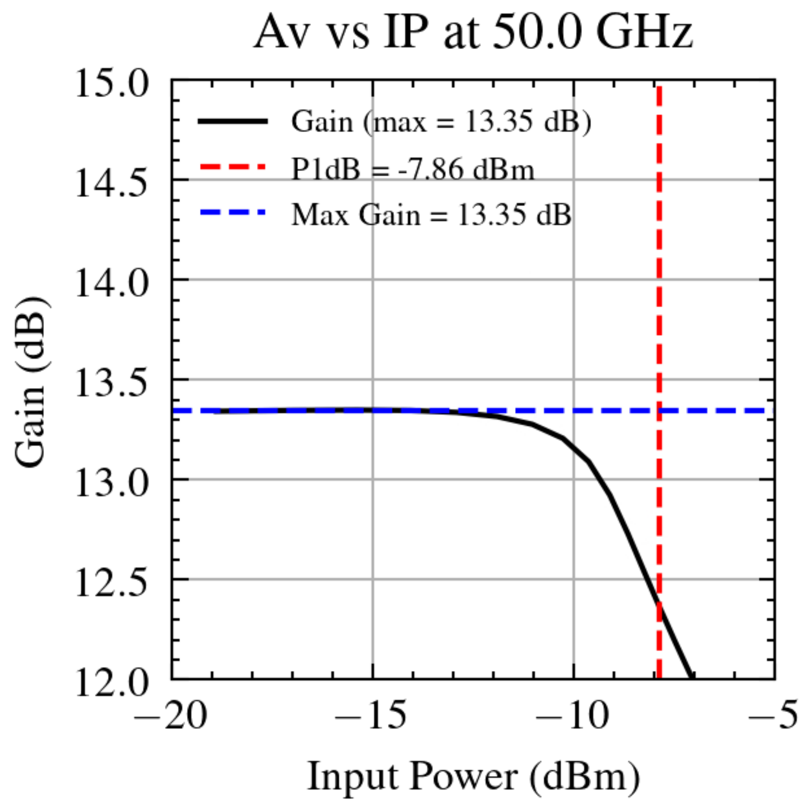
```python
# Import libraries
import pandas as pd
pd.set_option('display.float_format', lambda x: '%.22f' % x)
import numpy as np
import matplotlib.pyplot as plt
import os
import scienceplots
plt.style.use(['science', 'ieee'])
# creates figs directory
output_dir = 'figs'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
```

```python
# Read CSV data (change paths accordingly)
Iin_csv = pd.read_csv('../schematic/compression_1/csv/IPR2.csv', delimiter=';', comment='#')
Iout_csv = pd.read_csv('../schematic/compression_1/csv/IPR3.csv', delimiter=';', comment='#')
Vin_csv = pd.read_csv('../schematic/compression_1/csv/Vin.csv', delimiter=';', comment='#')
Vout_csv = pd.read_csv('../schematic/compression_1/csv/Vout.csv', delimiter=';', comment='#')

# Initialize an empty DataFrame with the correct number of rows based on Iin_csv
power_df = pd.DataFrame({
    'Freq': Iin_csv['FREQ'],
    'P_in': np.zeros(len(Iin_csv)),  # Preallocate with zeros
    'P_out': np.zeros(len(Iin_csv))  # Preallocate with zeros
})
```

```python
# Create complex voltage and current arrays
vin_complex = []
vout_complex = []
Iout_complex = []
Iin_complex = []

# Populate the complex lists
for i in range(len(Vin_csv['r xyce/V(VIN)'])):
    vin_complex.append(complex(Vin_csv['r xyce/V(VIN)'][i], Vin_csv['i xyce/V(VIN)'][i]))
    vout_complex.append(complex(Vout_csv['r xyce/V(VOUT)'][i], Vout_csv['i xyce/V(VOUT)'][i]))
    Iout_complex.append(complex(Iout_csv['r xyce/I(PR3)'][i], Iout_csv['i xyce/I(PR3)'][i]))
    Iin_complex.append(complex(Iin_csv['r xyce/I(PR2)'][i], Iin_csv['i xyce/I(PR2)'][i]))
```

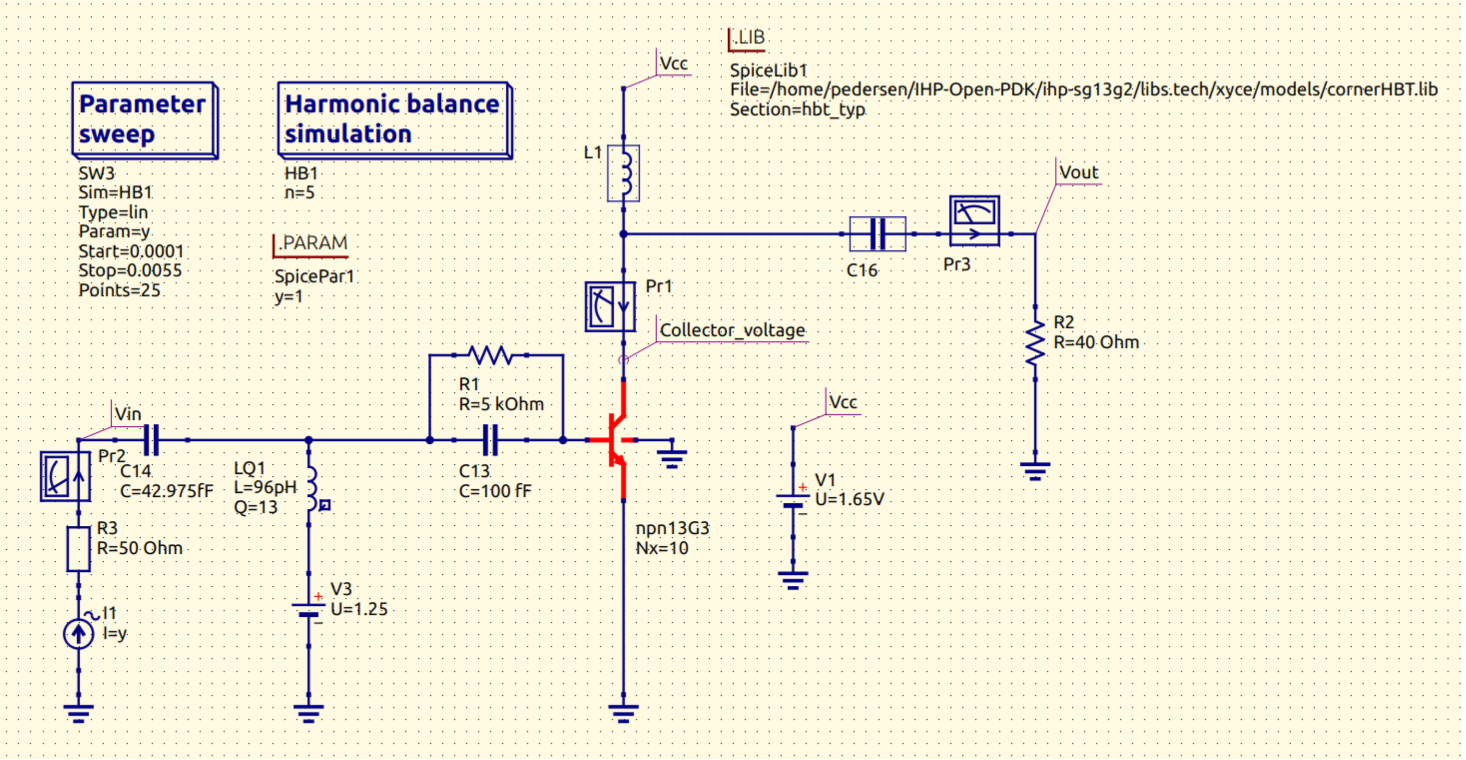
Visualizing The Results





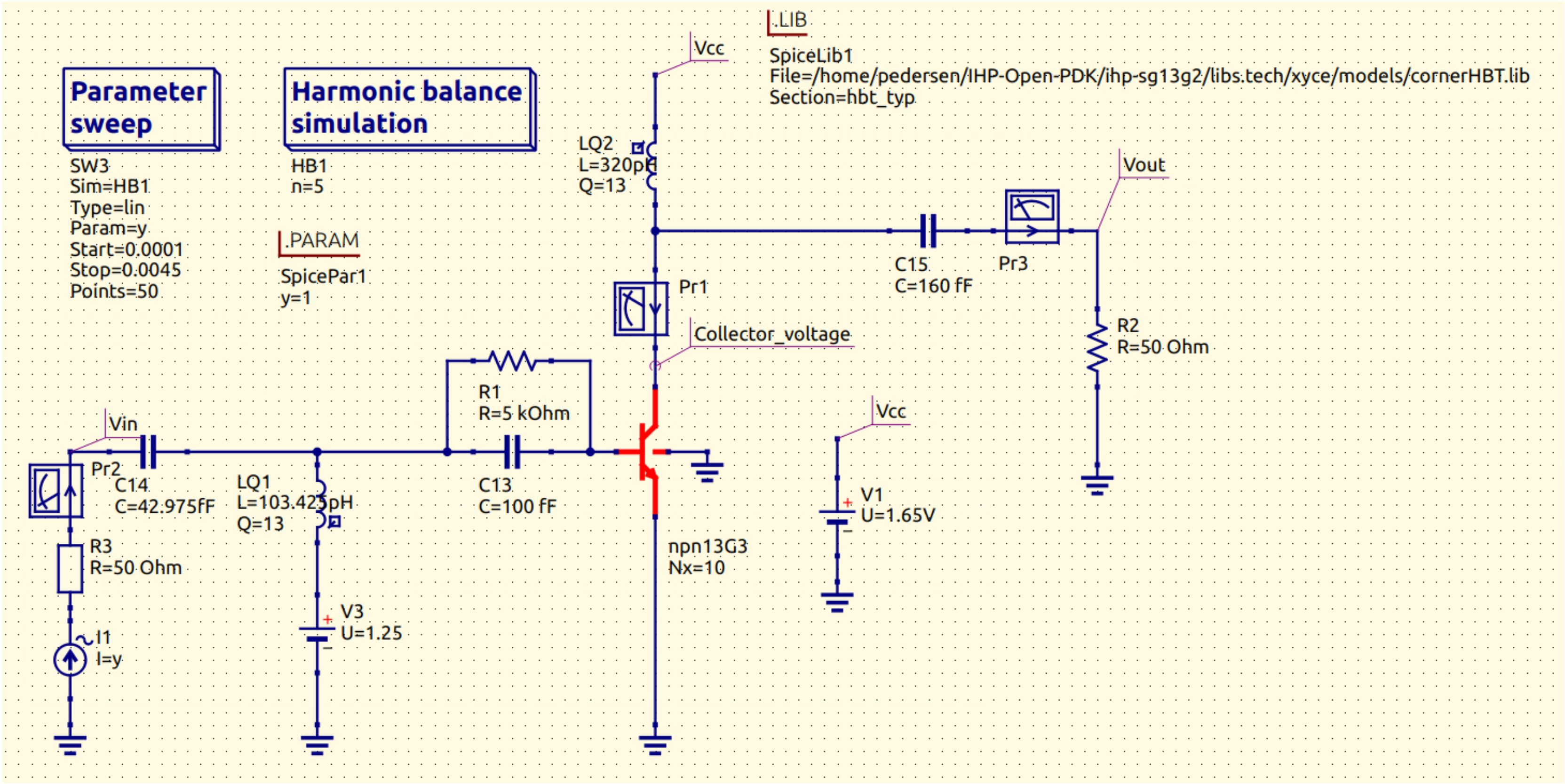
Load Pull

- The goal of the load-pull analysis is to optimize the **output network** for maximum **linearity**
- Explored a straightforward approach for multi-parameter sweeps in Xyce, but faced challenges due to limited documentation and unexpected complexities.
- Current simple approach: vary resistance and evaluate performance around 50 Ω.



Load Pull

- The schematic to the right is the one we will continue with..



Coffee Break!! / Catching Up

Leibniz Institute for high performance microelectronics

Part 4

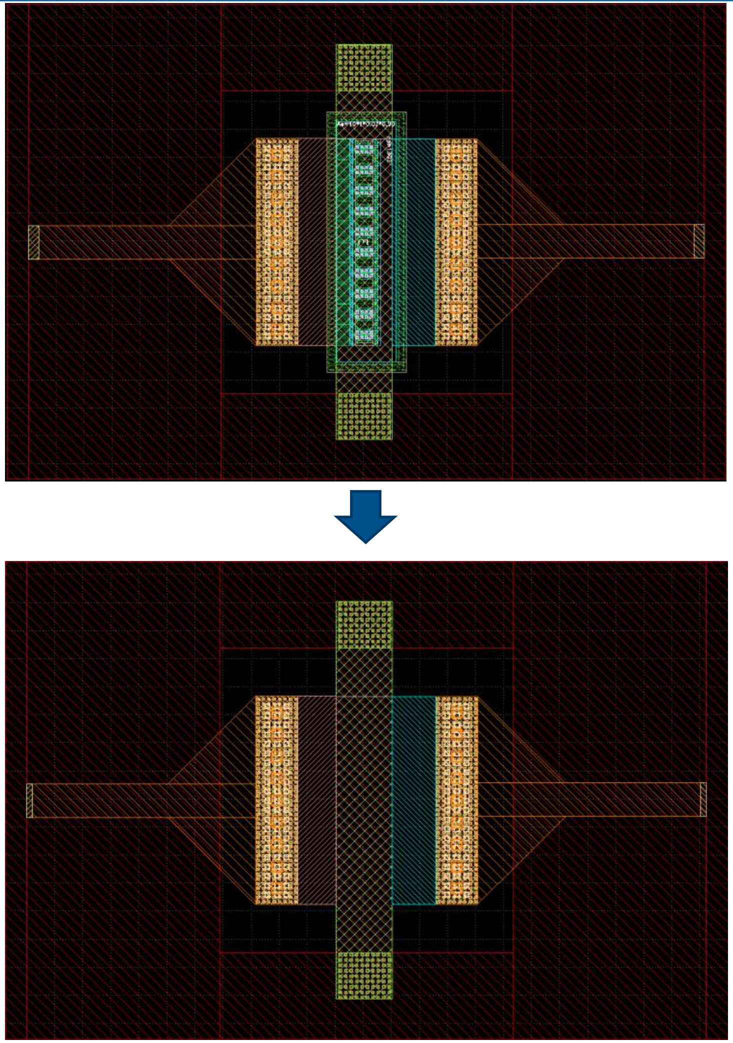
OpenEMS

Reference: modules/module_2_50GHz_MPA/part_4_layout_EMsims

Leibniz Institute for high performance microelectronics

EM Simulation

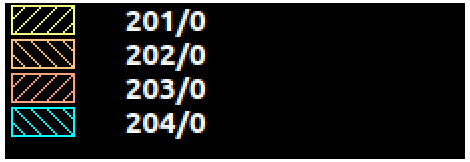
- At this point we want to include the EM simulated components into QUCS-S for post processing!
- We will use the approach proposed by **Dr.-Ing Volker Mühlhaus**

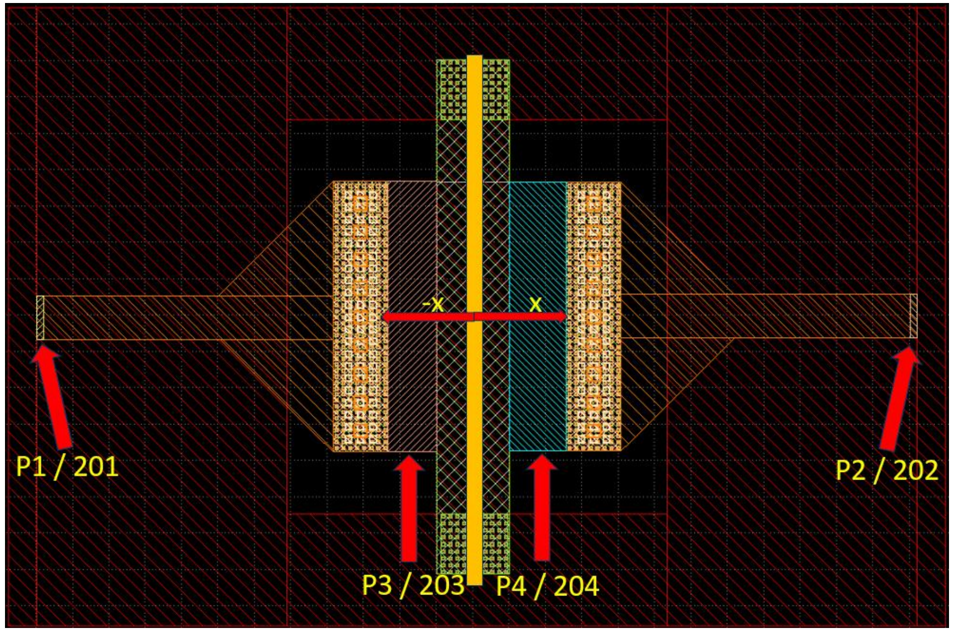


EM Simulation

- Define Input Ports on following layers
- The input ports should be referenced as described in the earlier presentation. For a detailed walkthrough of the code for this specific structure, refer to the following markdown file:

module_2_50GHz_MPA/part_4_layout_EMsims/EM_simulation.md





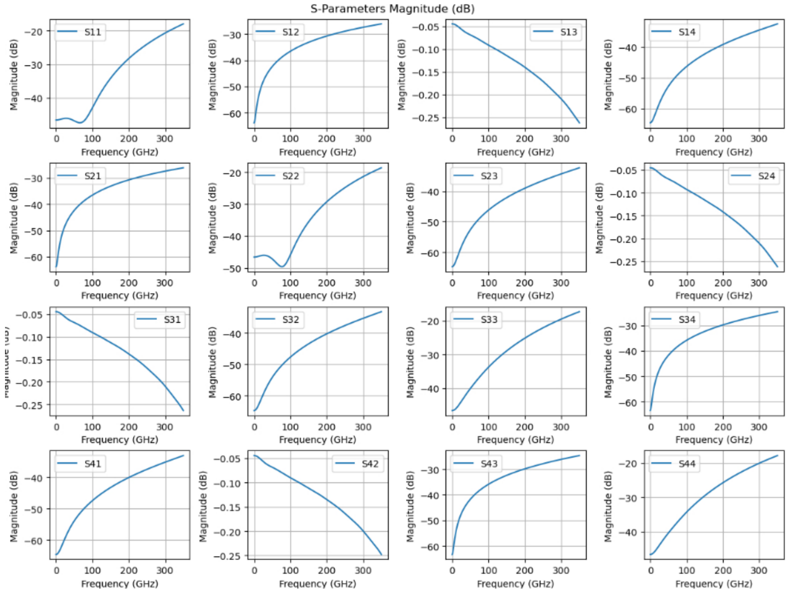
Post EM-Simulations

Viewing the S-Parameters:

openems/output/run_core_50ghz_mpa_data/run_core_50ghz_mpa_data

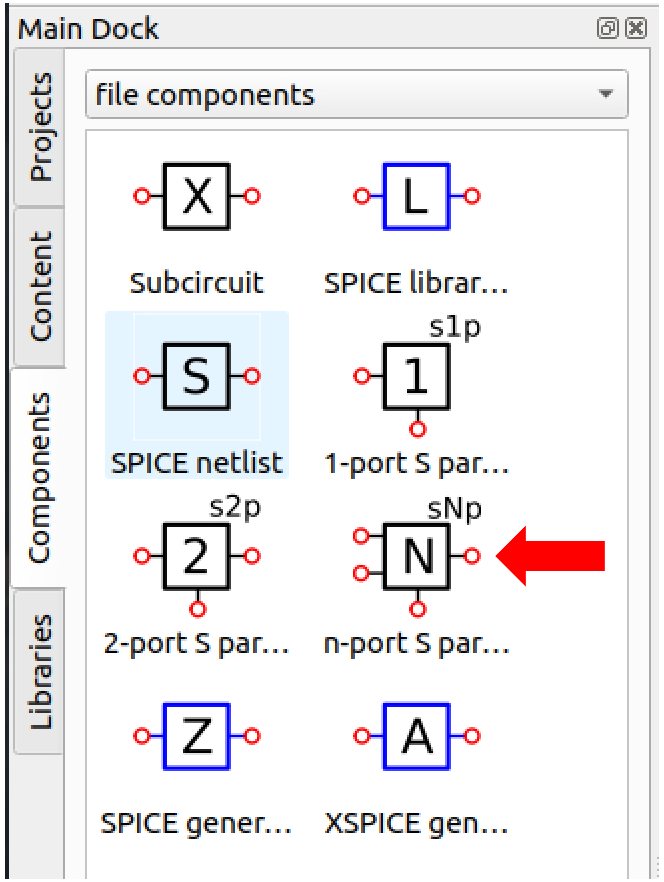
openems /output/run_core_50ghz_mpa_data/spar_plot.py **(this can be used for plotting the S-parameters)**

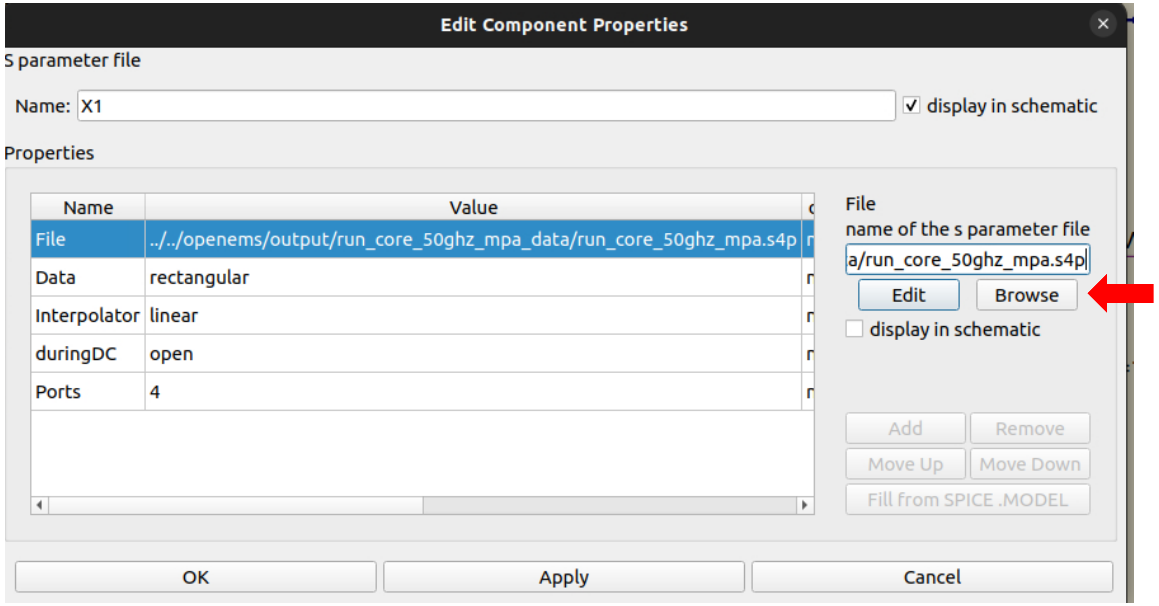
```
python3 spar_plot.py your_spar_file.s4p
```



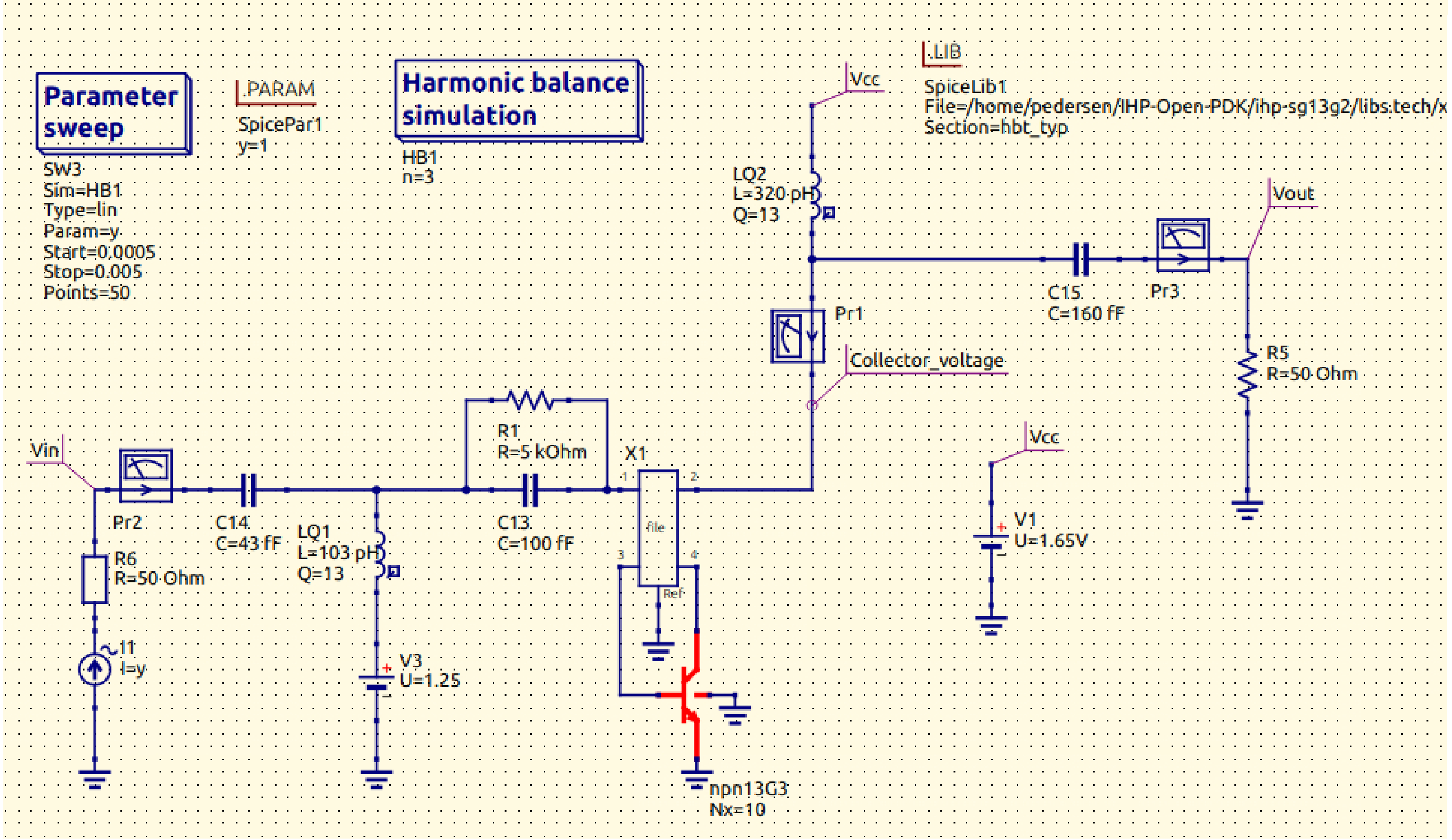
Post EM-Simulations

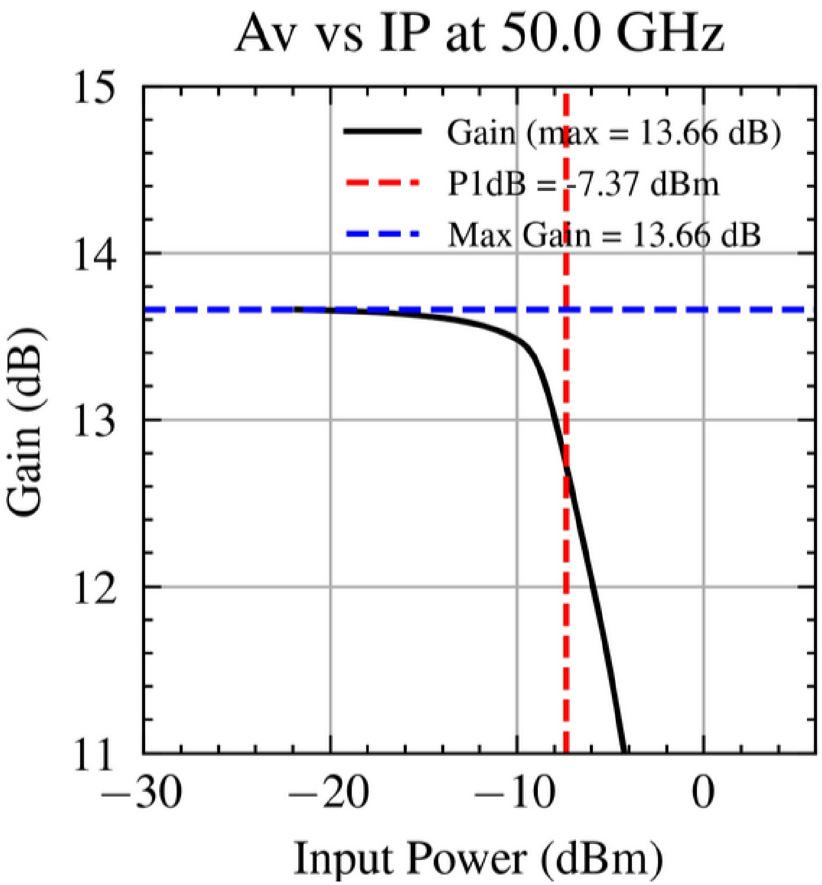
- Importing The EM model into QUCS-S:



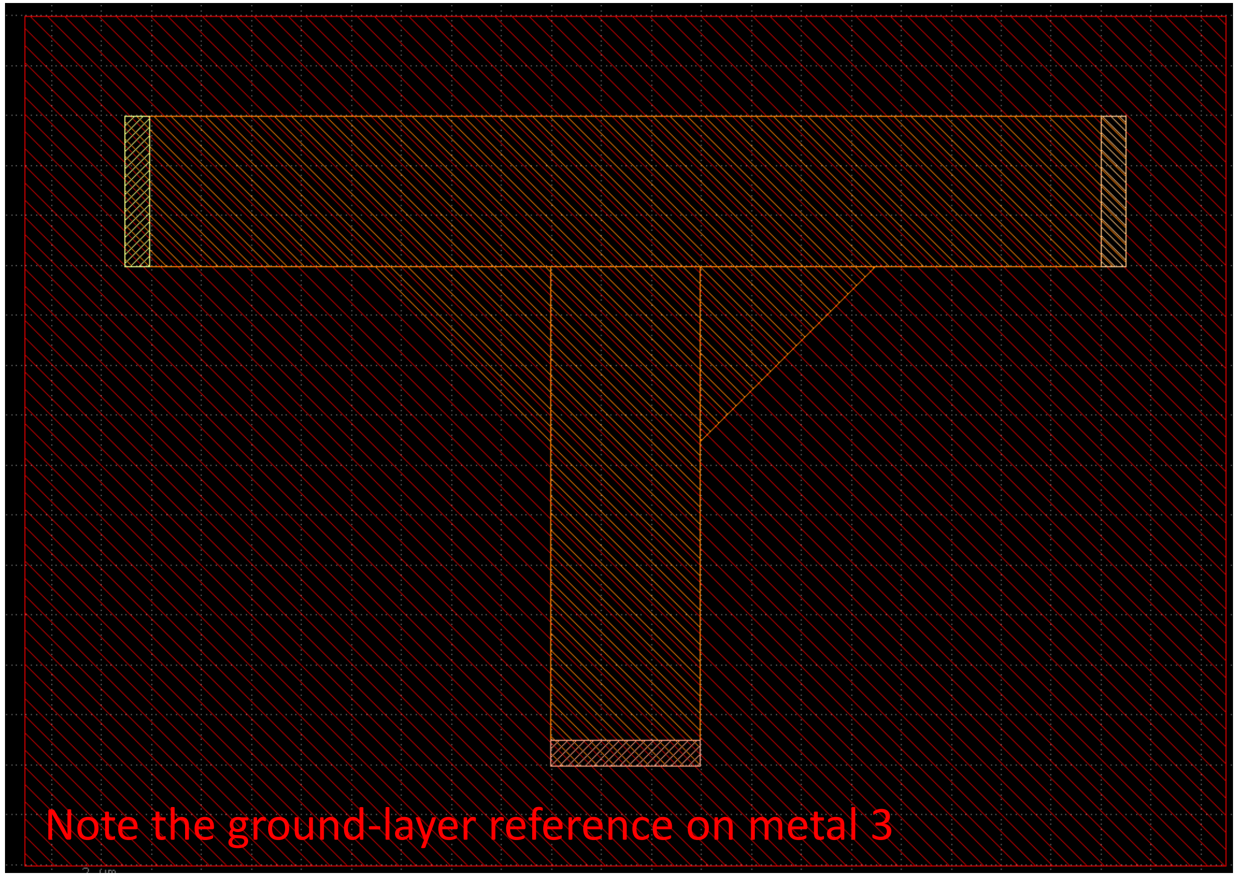


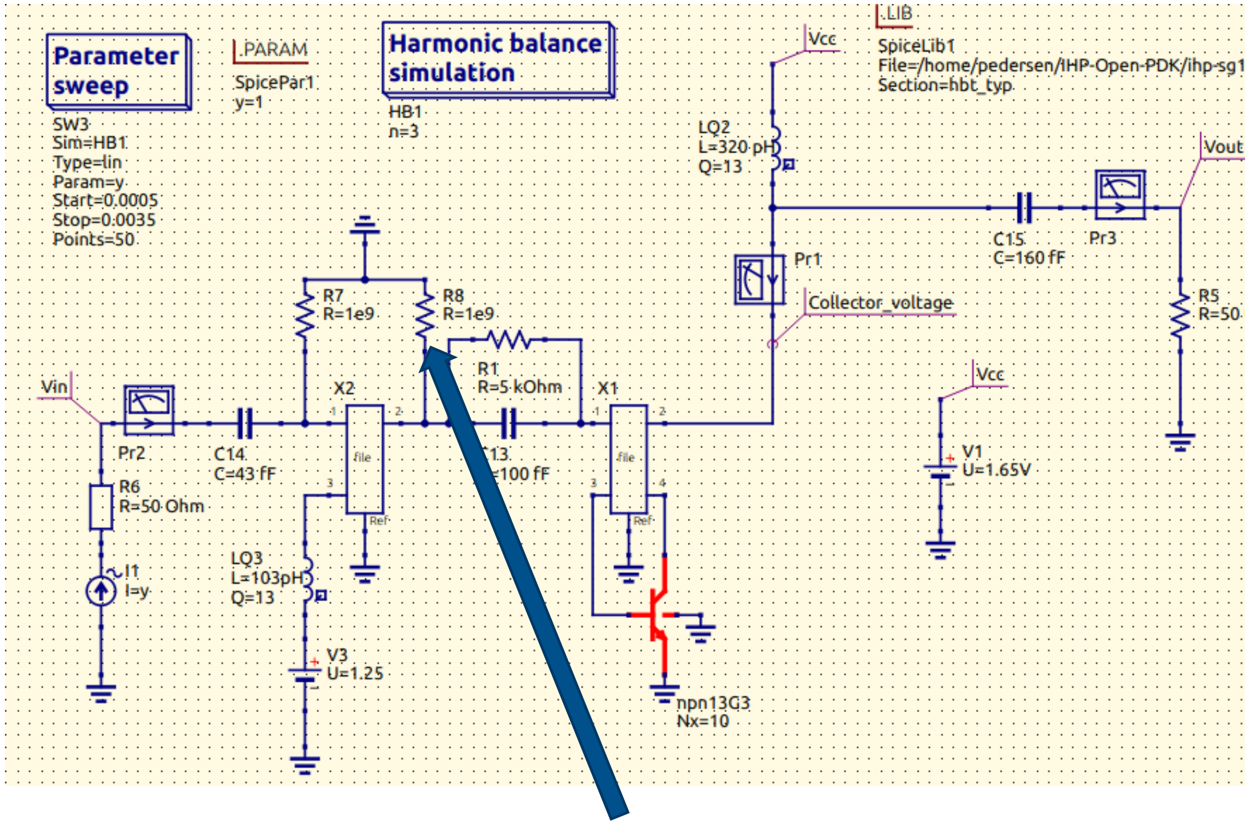
Post EM-Simulations





Post EM-Simulations





Used to set DC reference for simulator…..

Remaining Agenda for Today

- **Make additional components to be EM simulated (inductors, T-connections etc.):** Use this change to work on a machine with the full tool flow and get help
- **Design Competition:** Work on your layout for the big design competition!!
- **Catch-Up:** Work on material from today or the other days to catch up or clarify points that wasn’t clear
- **Relax ☺**

What Will We Do Tomorrow?

- **Xschem Practice:** Create small building blocks for a larger circuit and perform analyses such as Monte Carlo simulations based on your prior experience
- **Verilog:** Write and simulate simple Verilog code using open-source tools.
- **Analog-Mixed-Signal Integration**: Learn to integrate digital blocks into analog designs within Xschem.
- **8-bit SAR ADC Simulation**: Build and simulate a basic 8-bit successive approximation register (SAR) ADC using the day's designs.
- **Post-Processing with Python**: Plot and analyze transient simulation results using Python.

Bonus Material

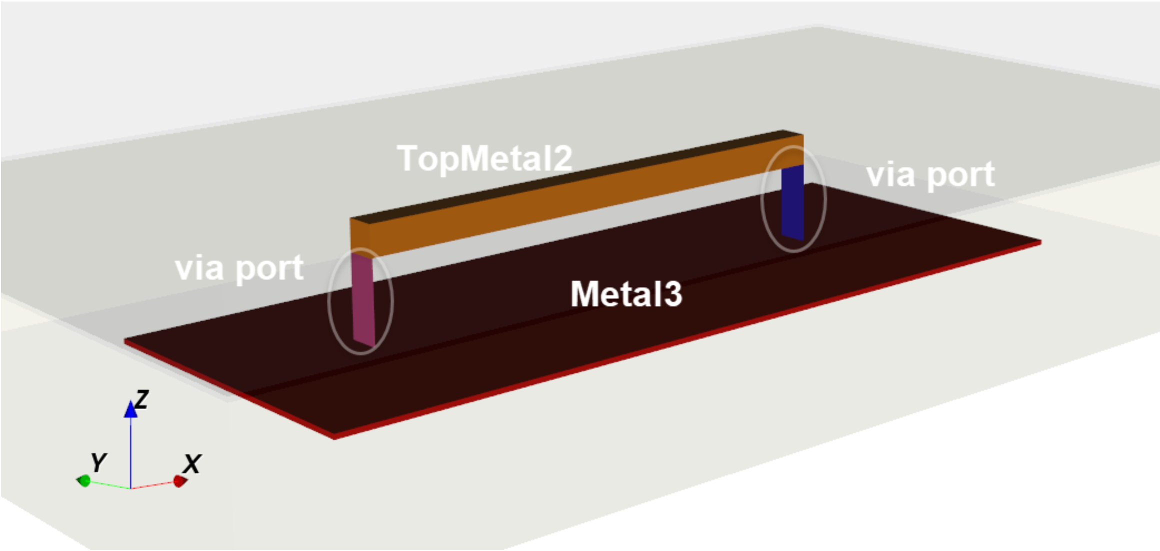
Port Parasitics in openEMS Models

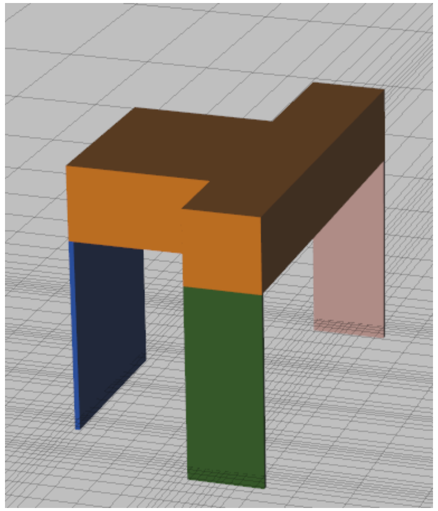
Volker Mühlhaus

13-May-2025

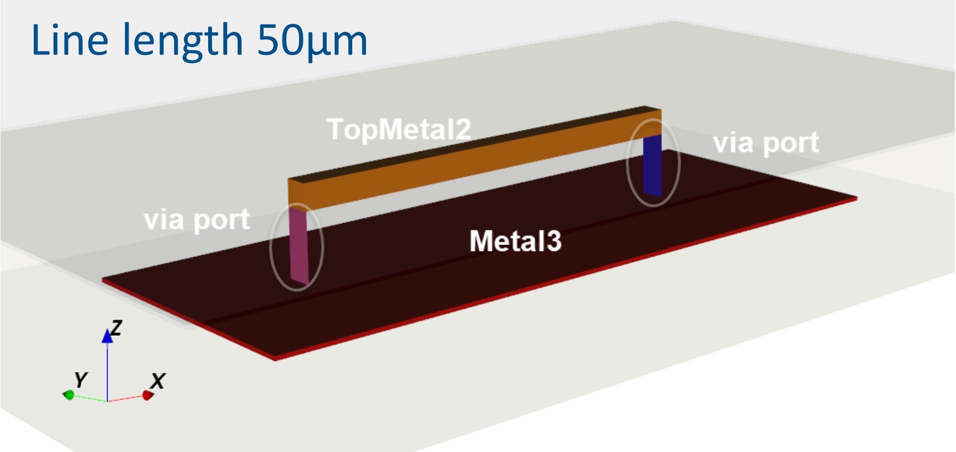
Effect of port size

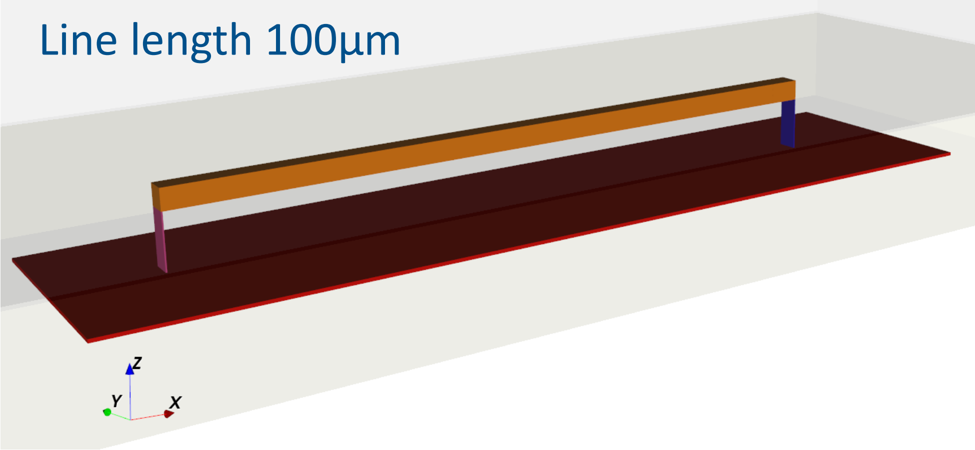
- Ideally, we want ports to be small compared to wavelength and DUT
- Sometimes this is not possible
- Ports then add physical length -> phase to results

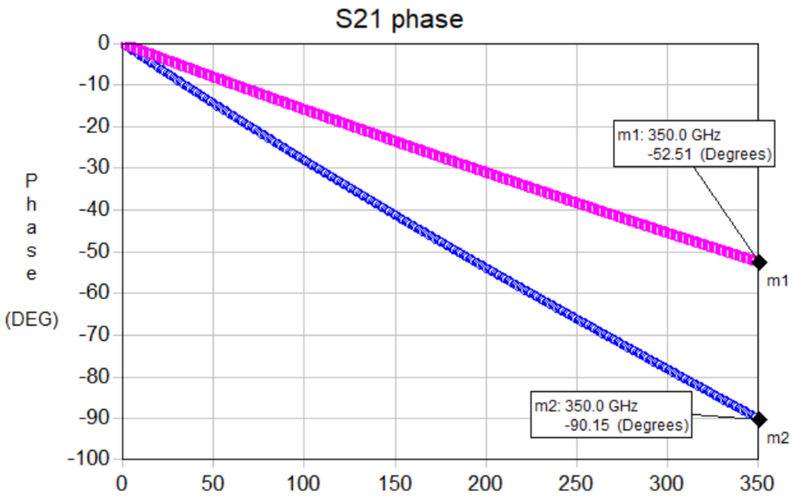




Test: Line 50µm and 100µm length







Not the expected double phase at double line length!

De-embedding software by Amphenol

https://www.amphenol-cs.com/software

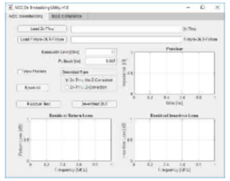
Amphenol
COMMUNICATIONS SOLUTIONS

CONNECTORS CABLES TRANSCEIVERS APPLICATIONS RESOURCES | CART | LOGIN | REGISTER

HOME | SOFTWARE

ACS DE-EMBEDDING UTILITY

In addition to iTRL, Amphenol is providing de-embedding software that uses 2x-thru and impedance corrected 2x-thru algorithms. The 2x-thru algorithm is fast, robust, and provides the same accuracy as TRL with only one input standard. Like iTRL, the impedance corrected 2x-thru algorithm corrects for differences between the test fixture and the 2x-thru to ensure causal de-embedded S-parameters. However, this utility only needs a 2x-thru instead of three line measurements. The software can de-embed single-ended and differential test fixtures, and also compares test fixture 2x-thru S-parameters to the IEEE P370 Fixture Electrical Requirements.

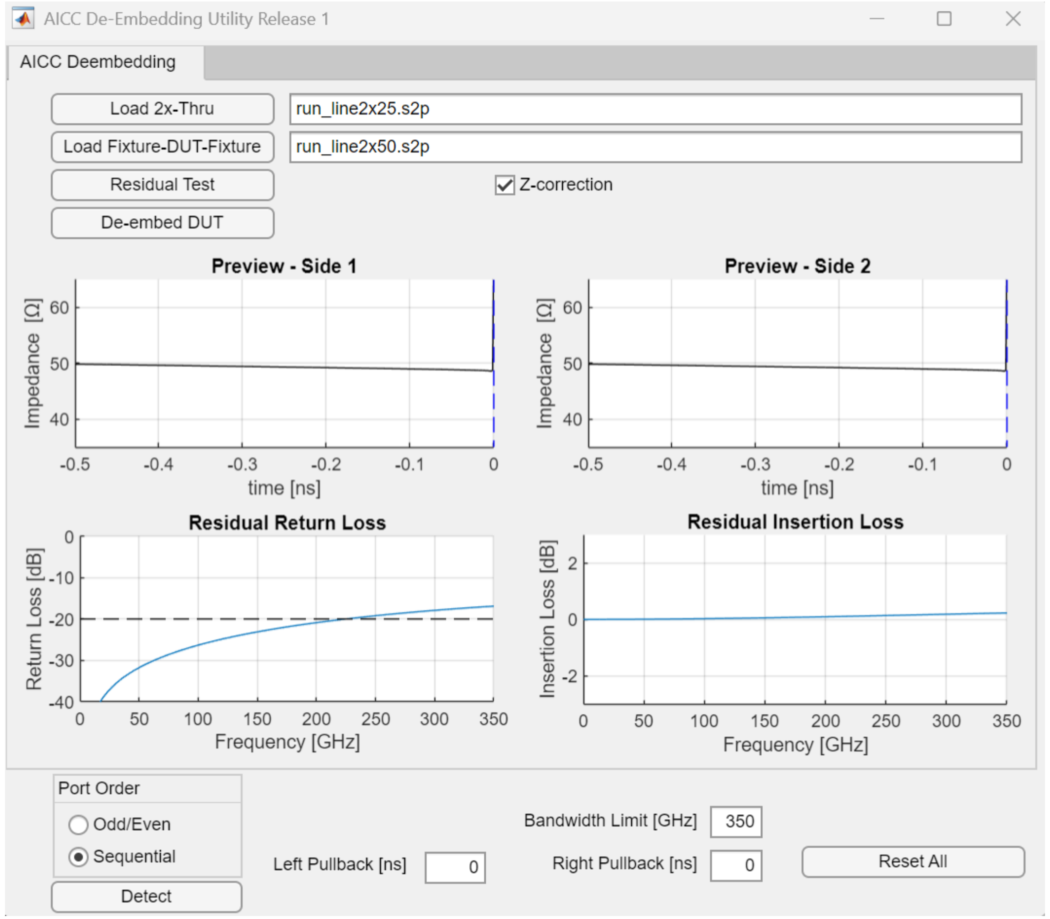


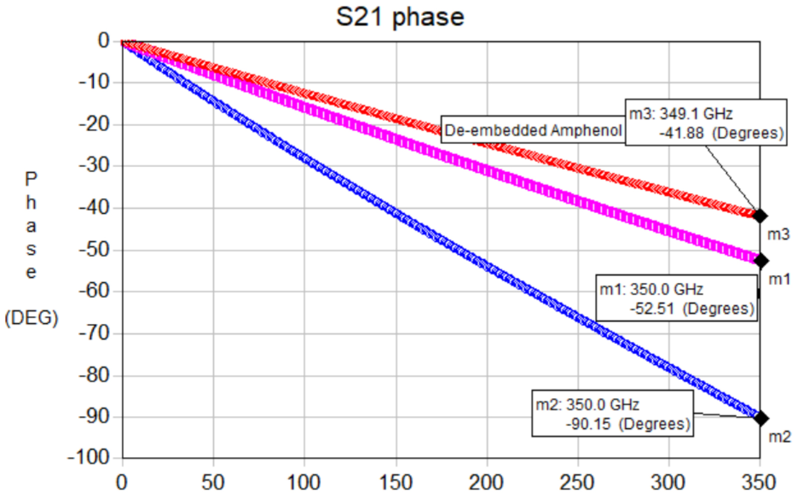
ACS De-embedding Utility (Zipped)

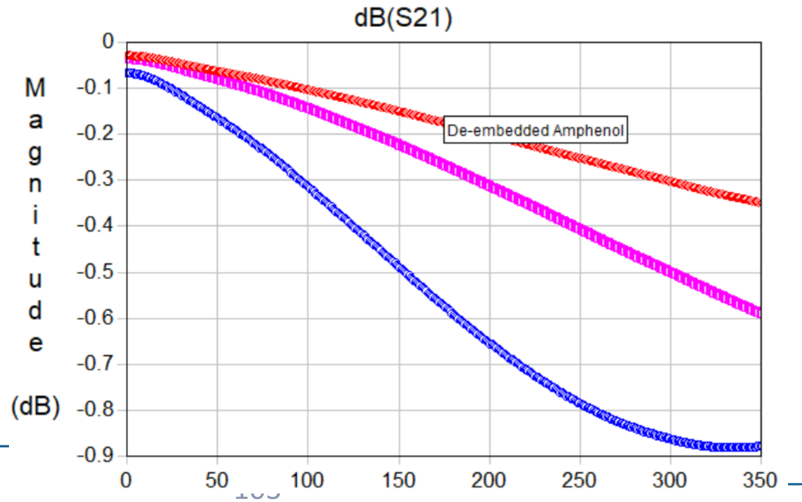
Beta Release - Software and Manual

Download

De-embedding software by Amphenol

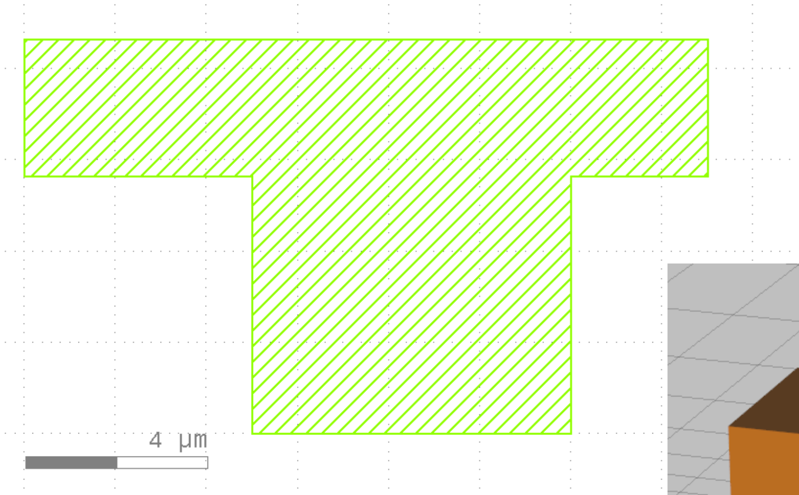


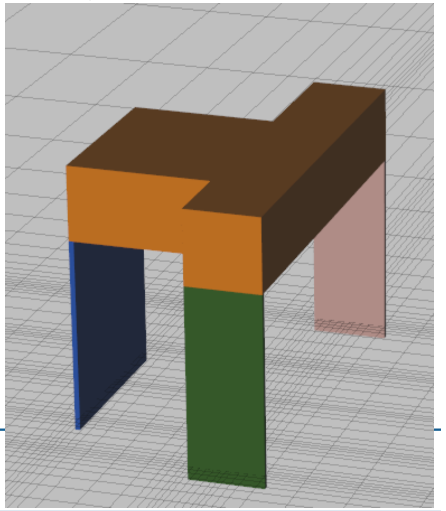


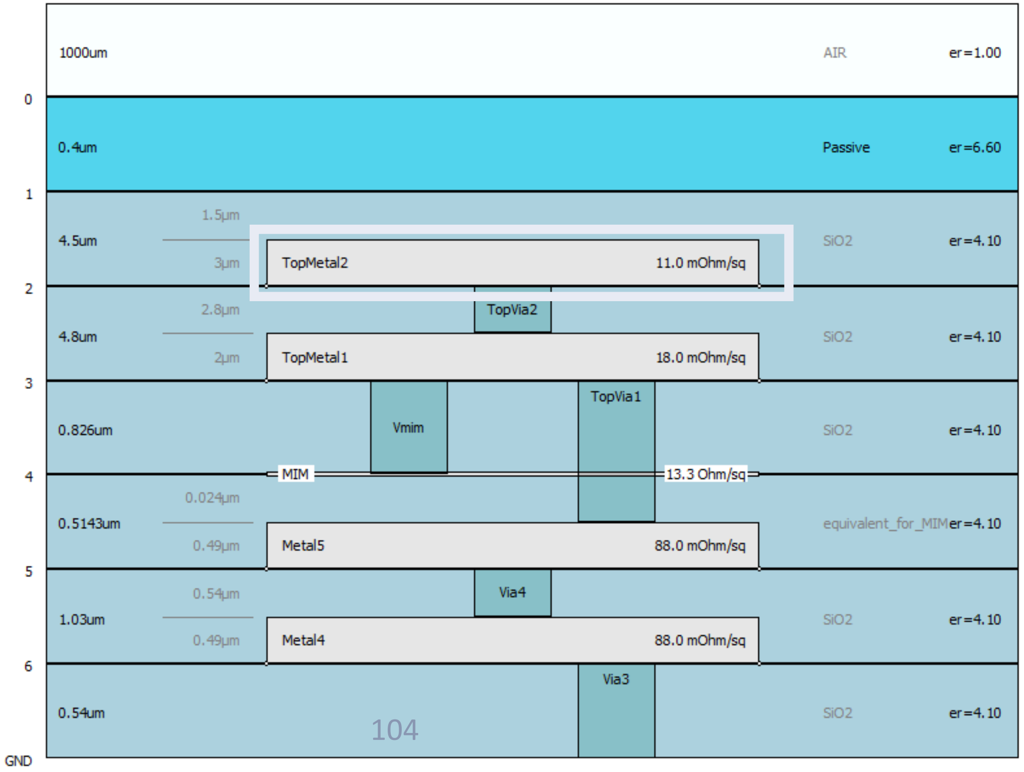


T-junction model, this is a small geometry

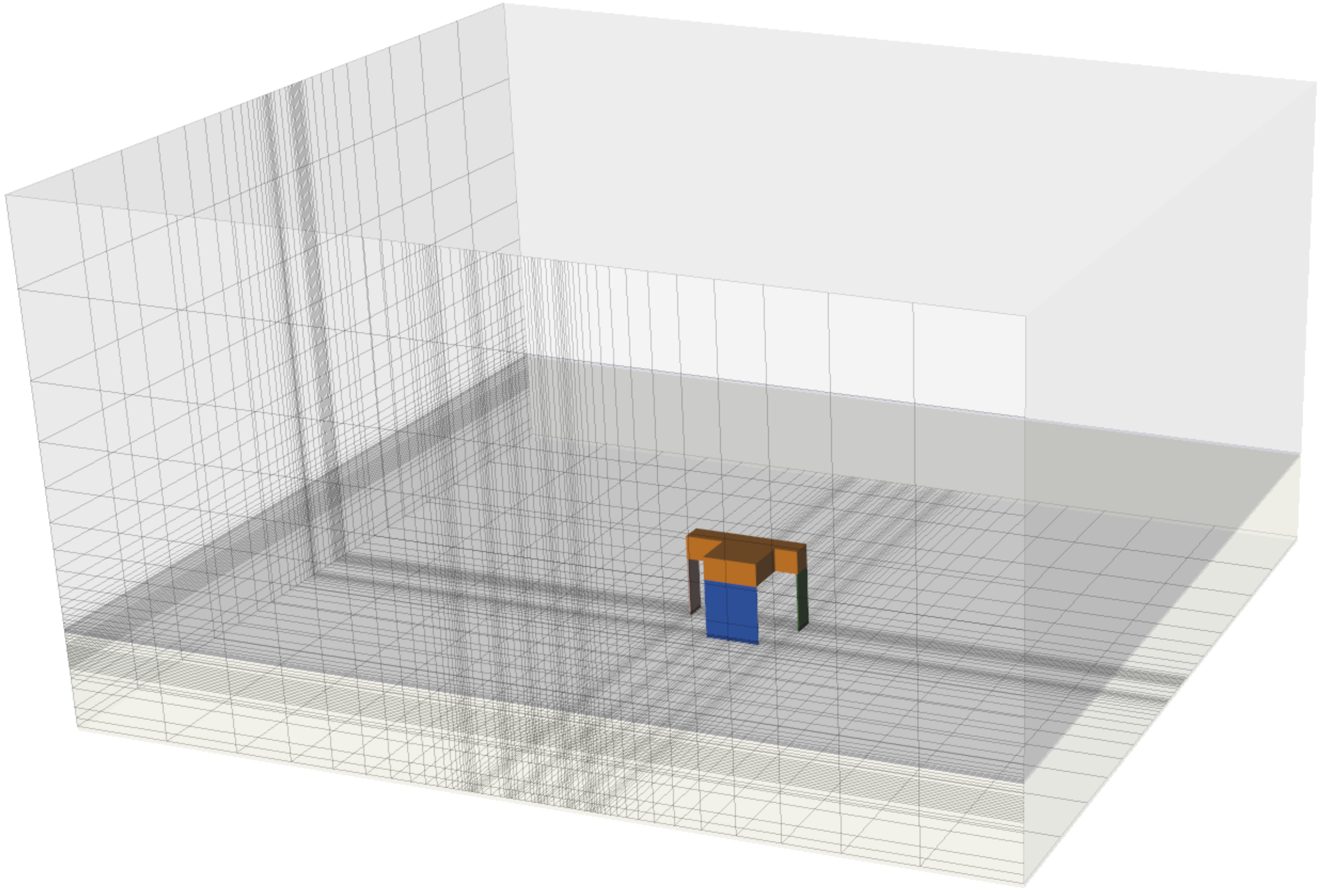
- TM2 over M3
- M3 replaced by simulation boundary (lossless) in this model





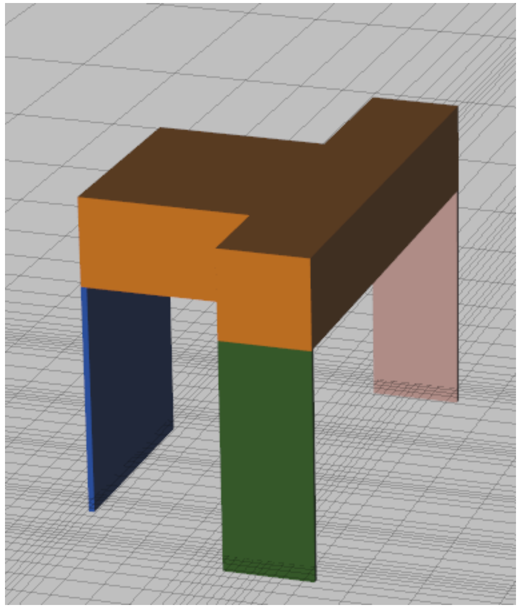


3D model view in openEMS viewer

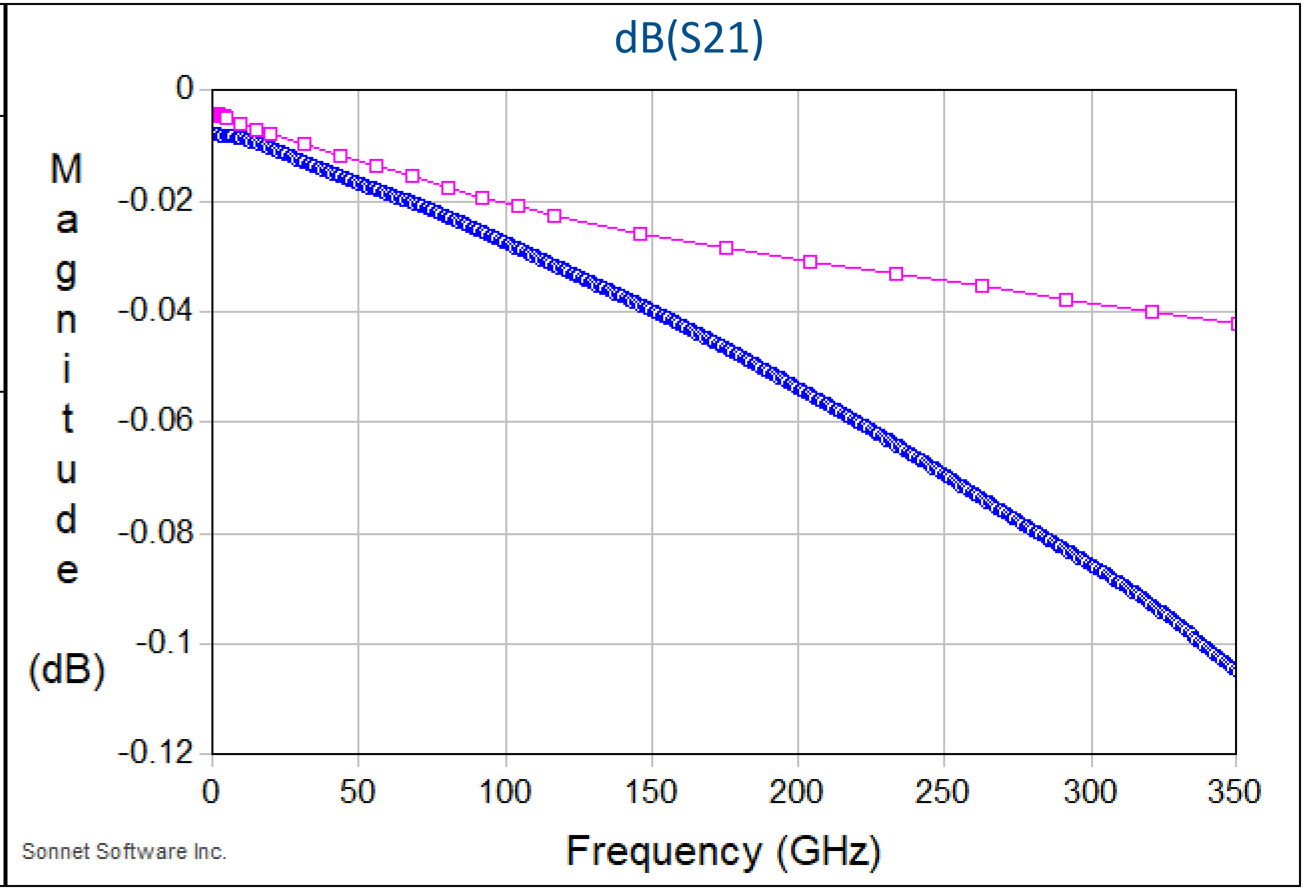


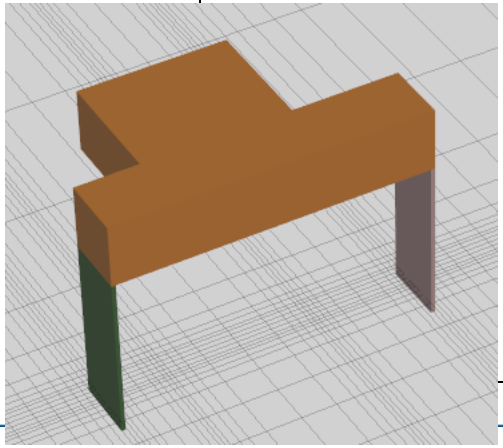
Test for effect of port size

- Port length from TM2 down to M3 is significant compared to DUT size
- Port modelled as via port here -> change in current direction at feed
- Need to check the effect introduced by the ports
- Testcase: reduce to two-port thru, remove port 3
  and check S21 magnitude and phase

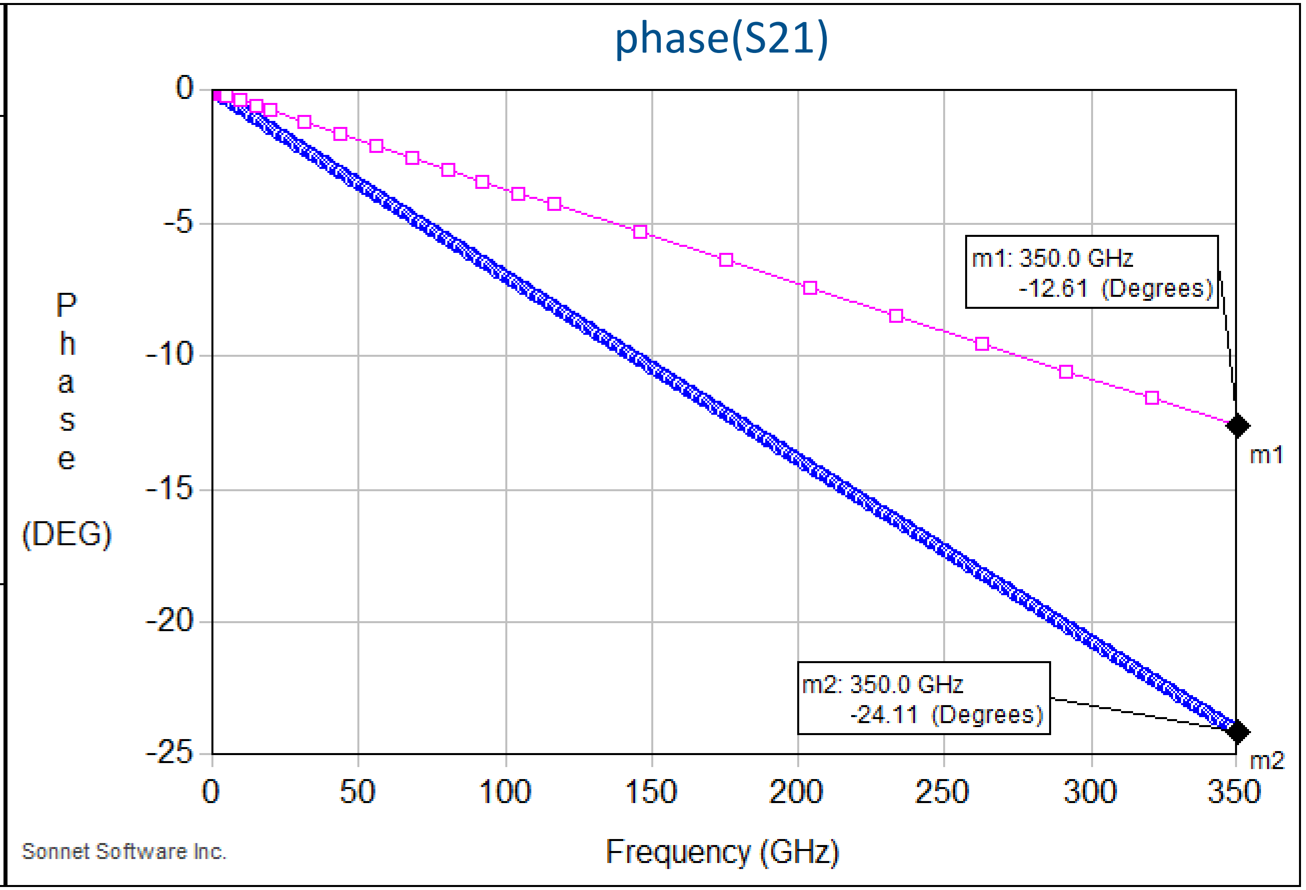


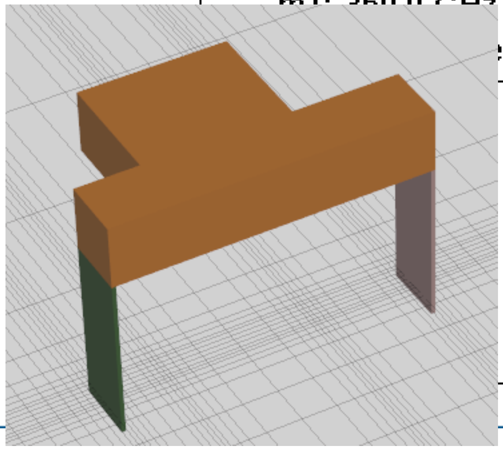
2-port testcase compared to other EM



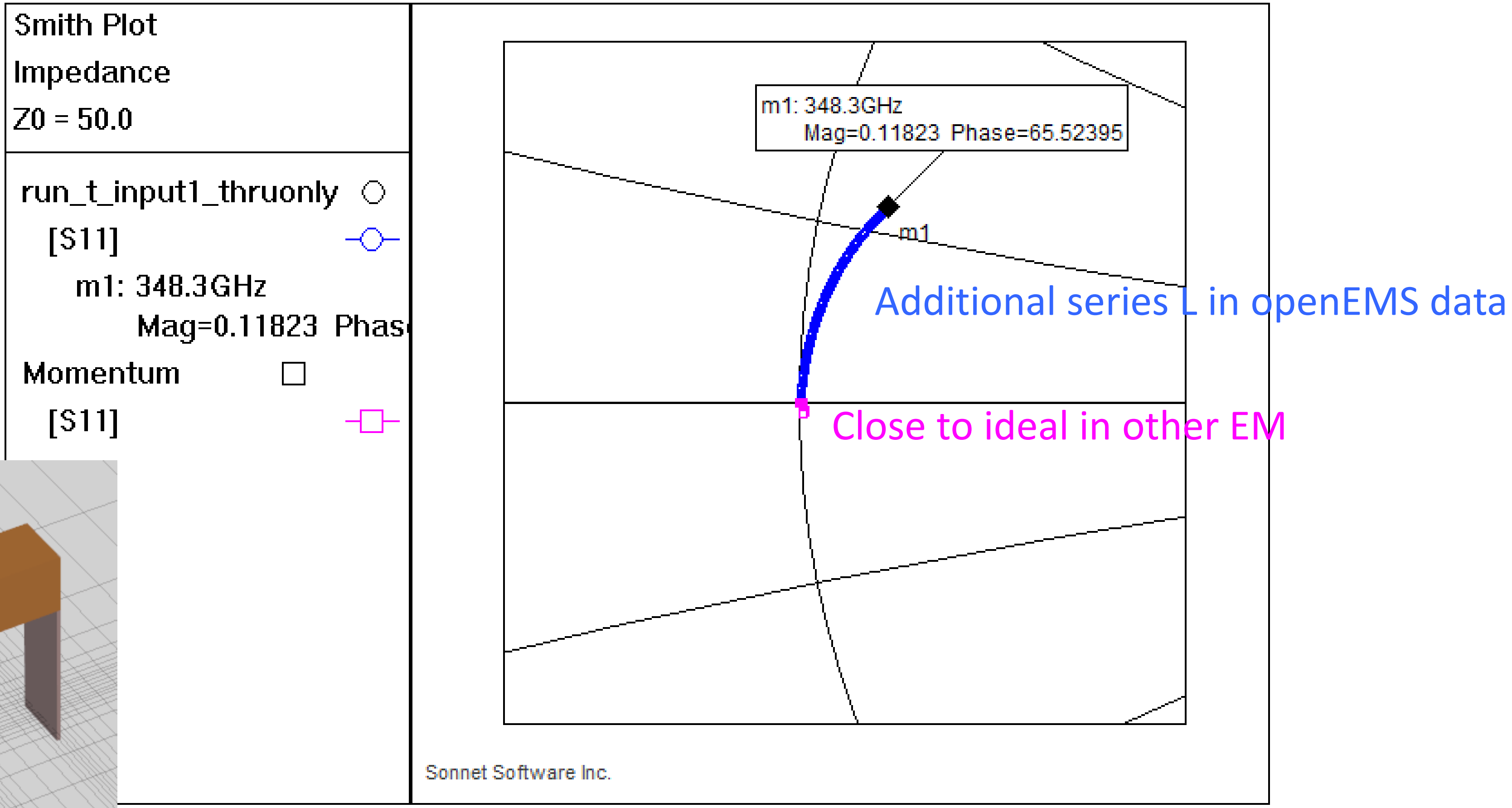


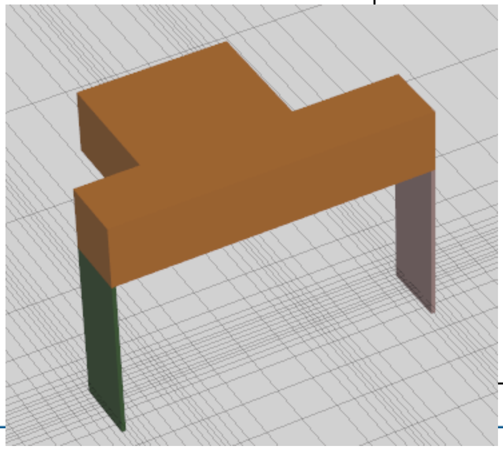
2-port testcase compared to other EM





2-port testcase compared to other EM

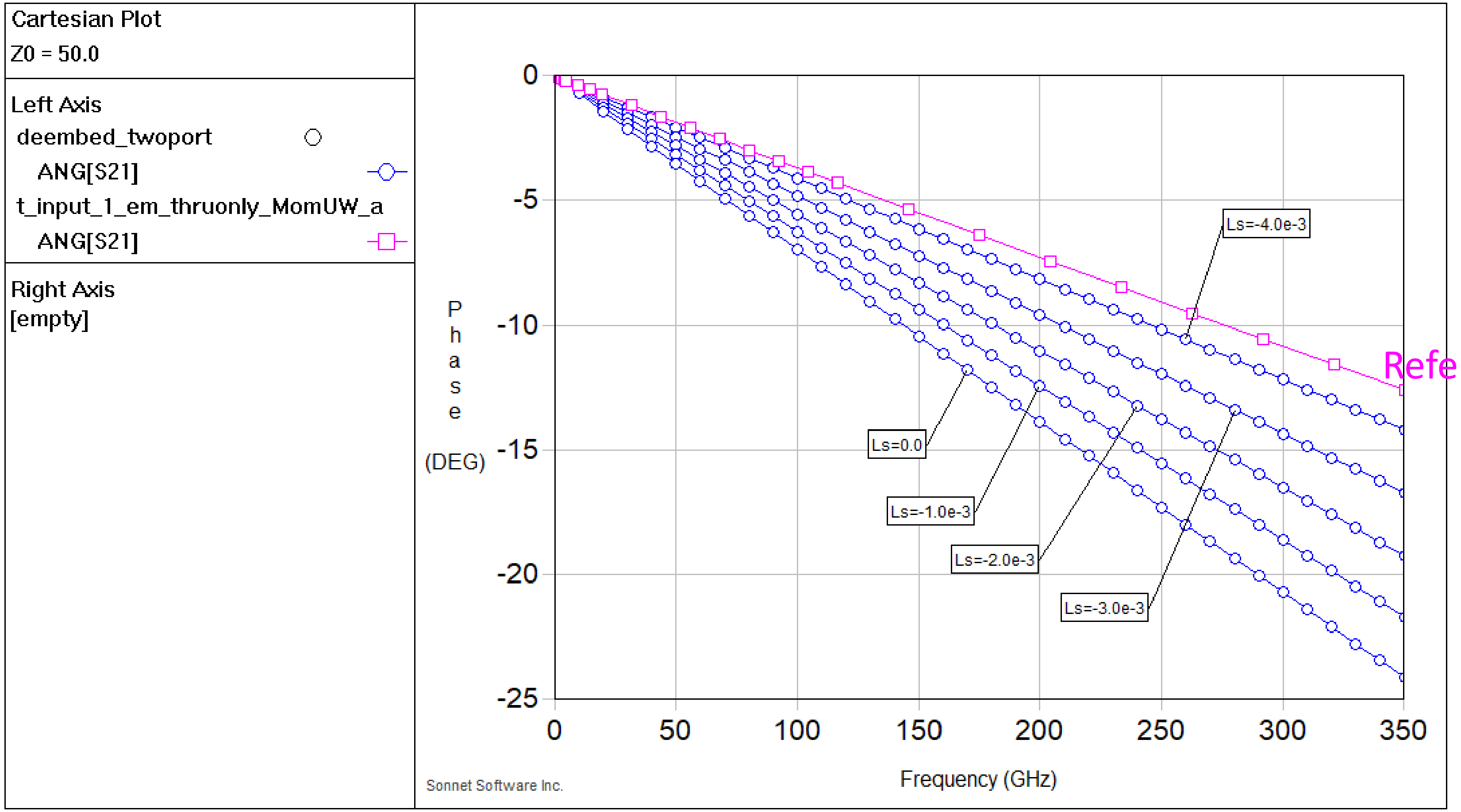




Conclusion T-junction model t_input_1

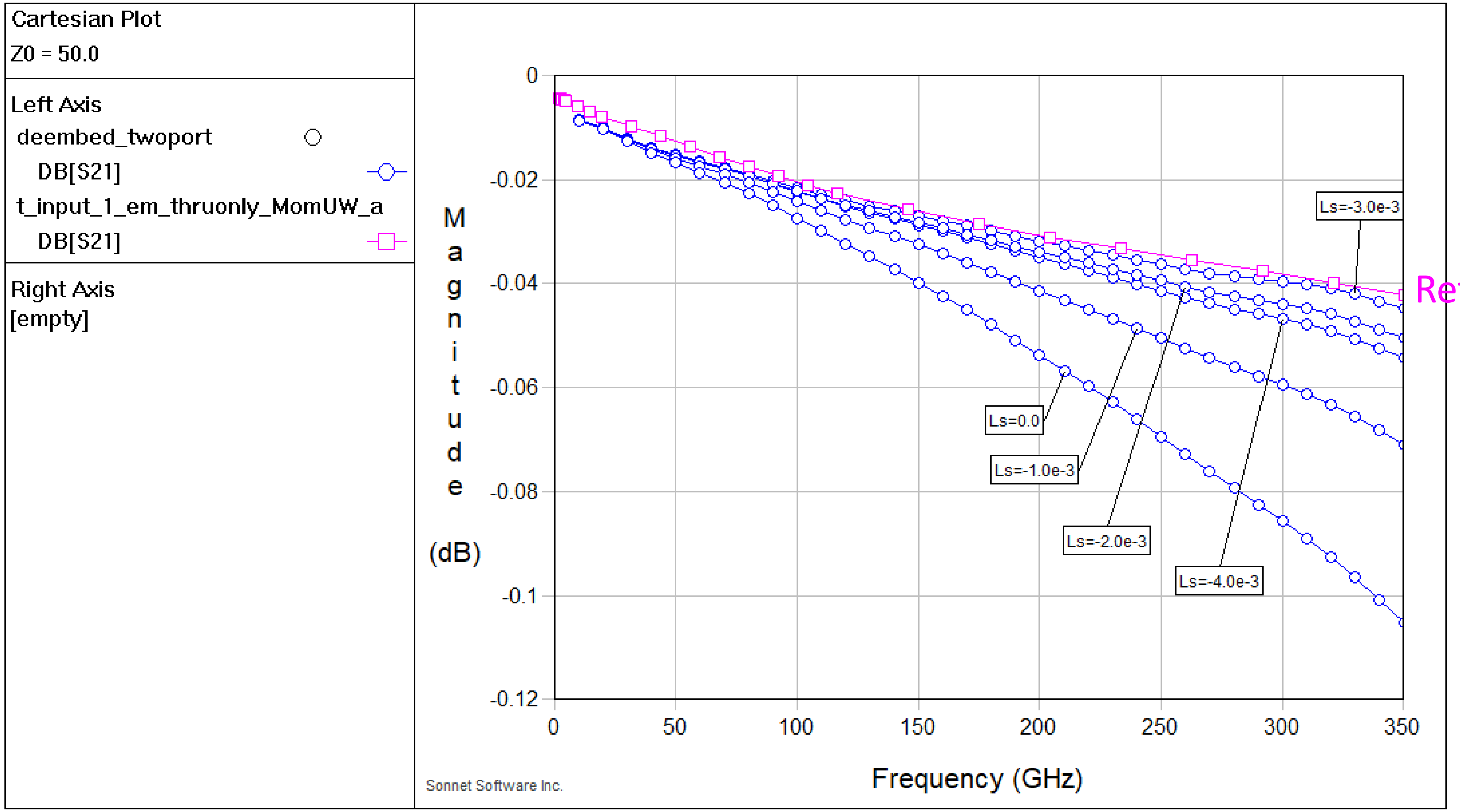
- Size of ports is here on the order of the small T-junction, adding inductance
- For such DUT on the order of port size, we try to compensate the effect of via ports **=> add negative inductance**

Phase(S21) after removing 0...4pH per port



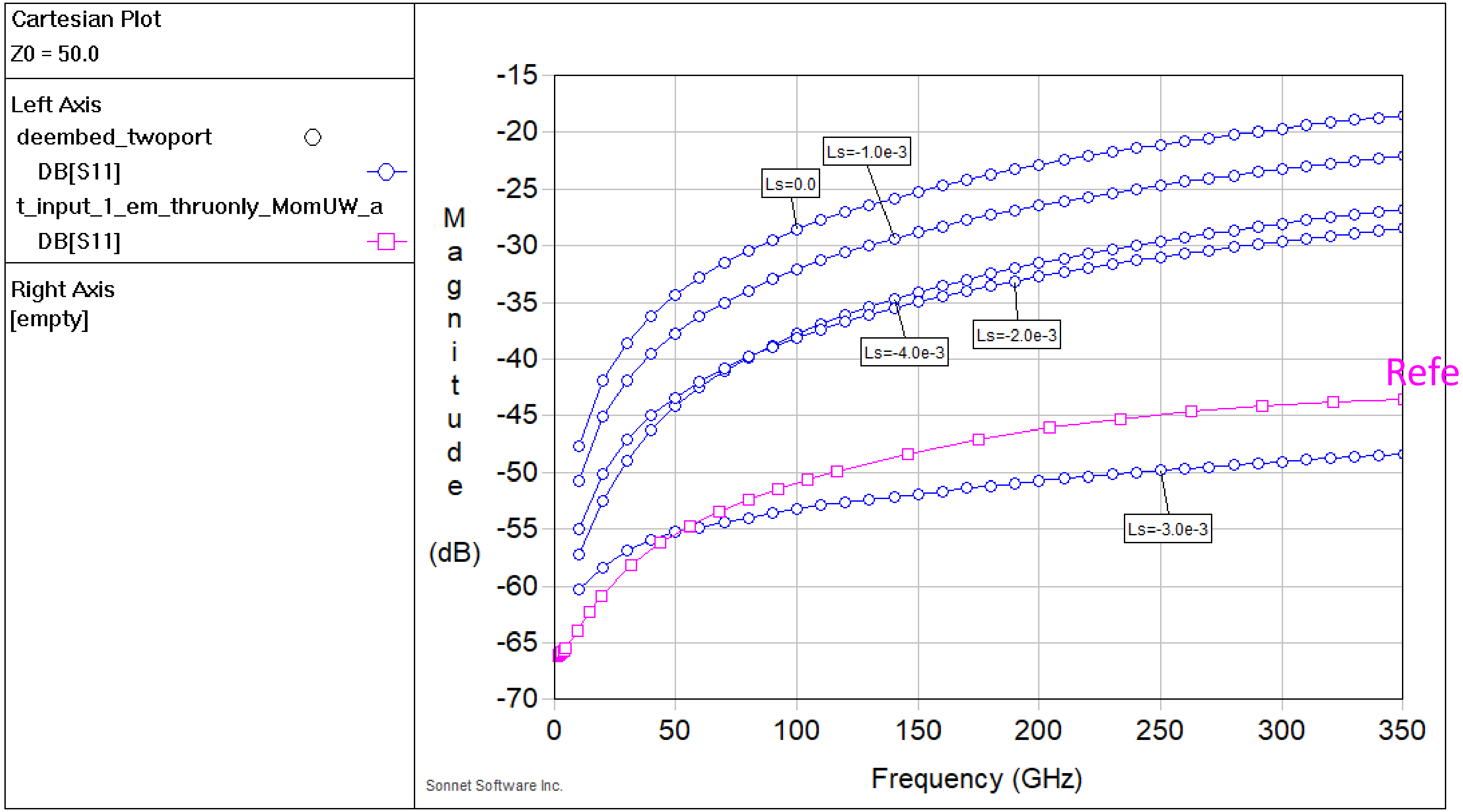
Reference: other EM

dB(S21) after removing 0...4pH per port



Reference: other EM

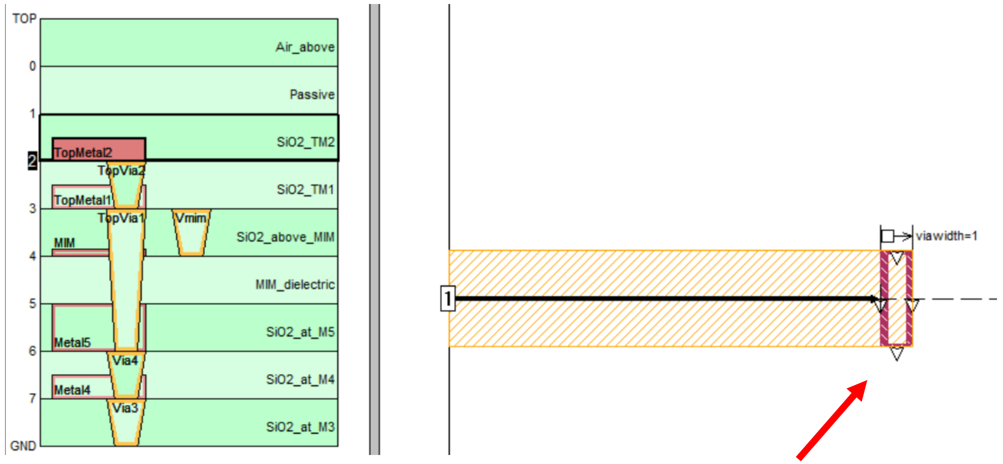
S11 after after removing 0...4pH per port



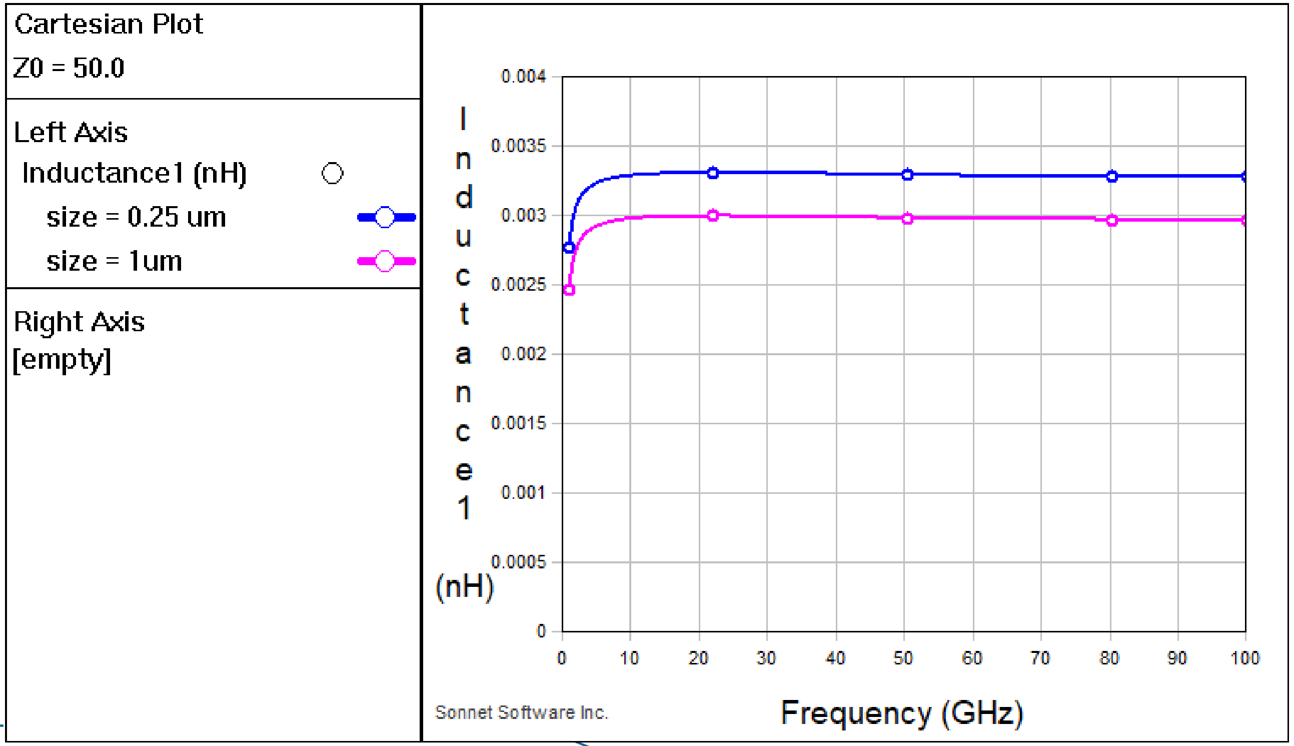
Reference: other EM

Double check: via of that size in Sonnet EM

- Simulate „via only“ in another EM tool with built-in port calibration
- Result is ~ 3pH, in agreement with previous findings



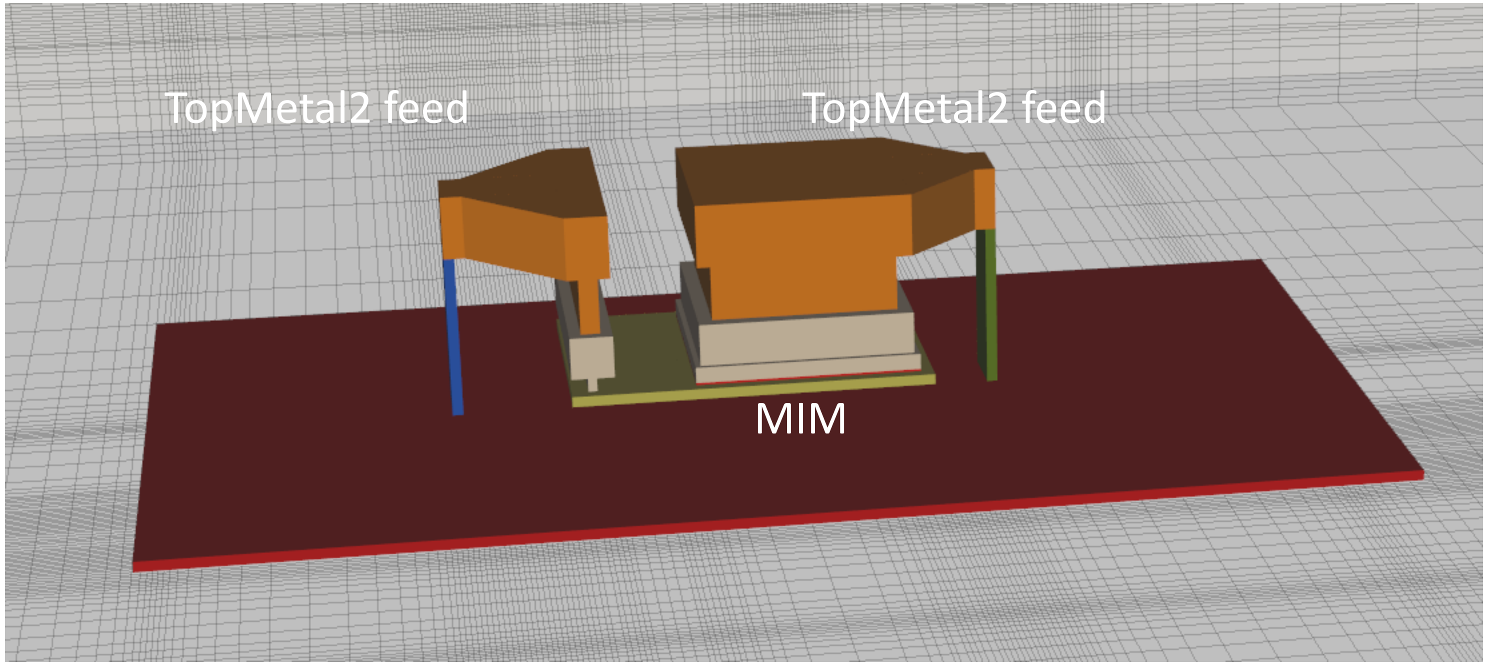
Via with same size as openEMS port



Applied to MIM model

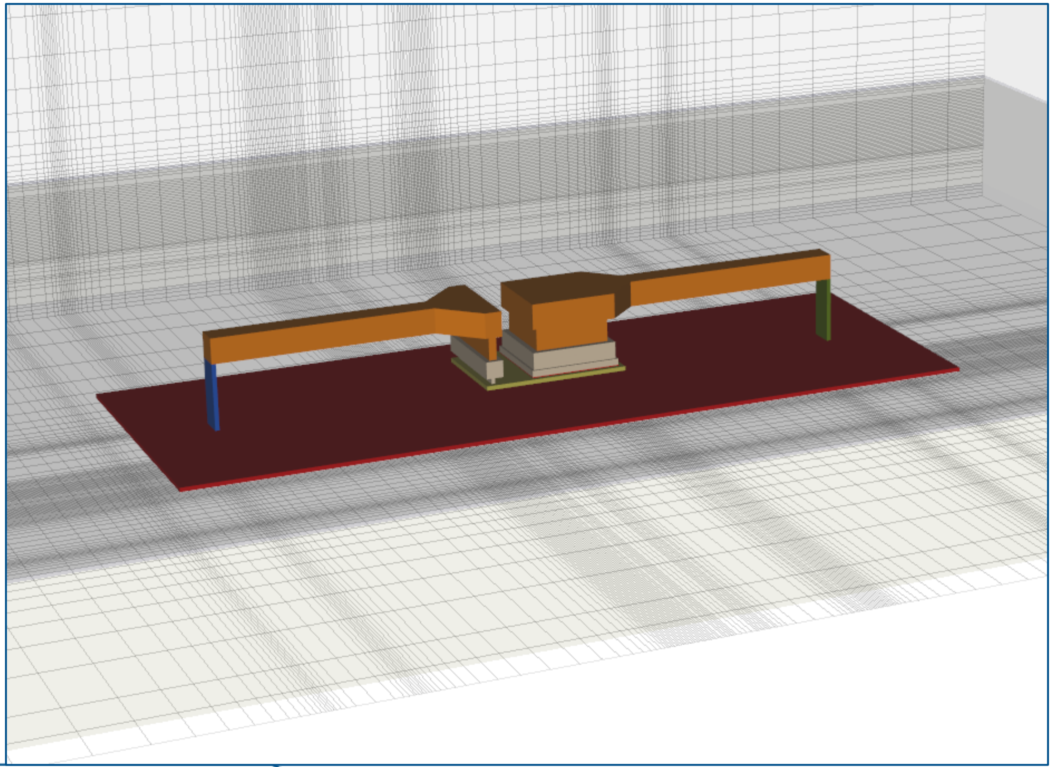
Testcase MIM

- Using via ports directly at the DUT, we would add too much parasitic effect, due to port length and coupling port <> DUT

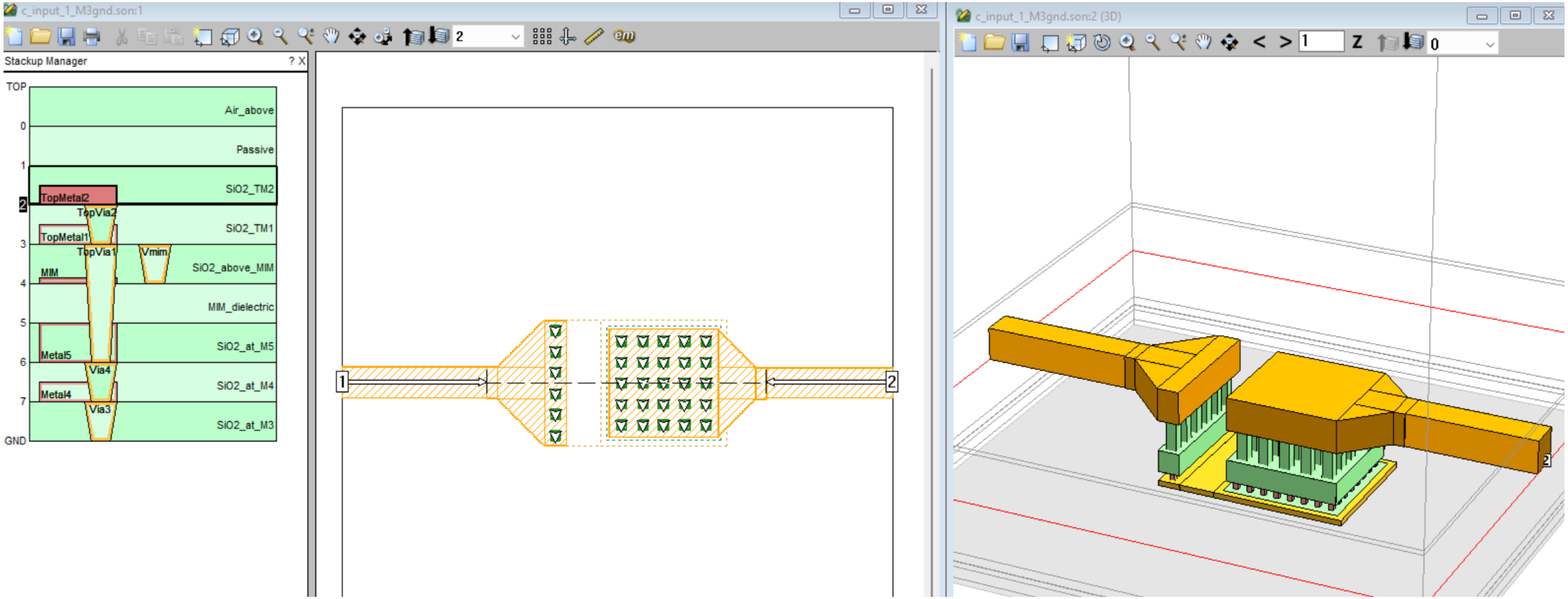


Idea: add feed line with via ports, de-embed

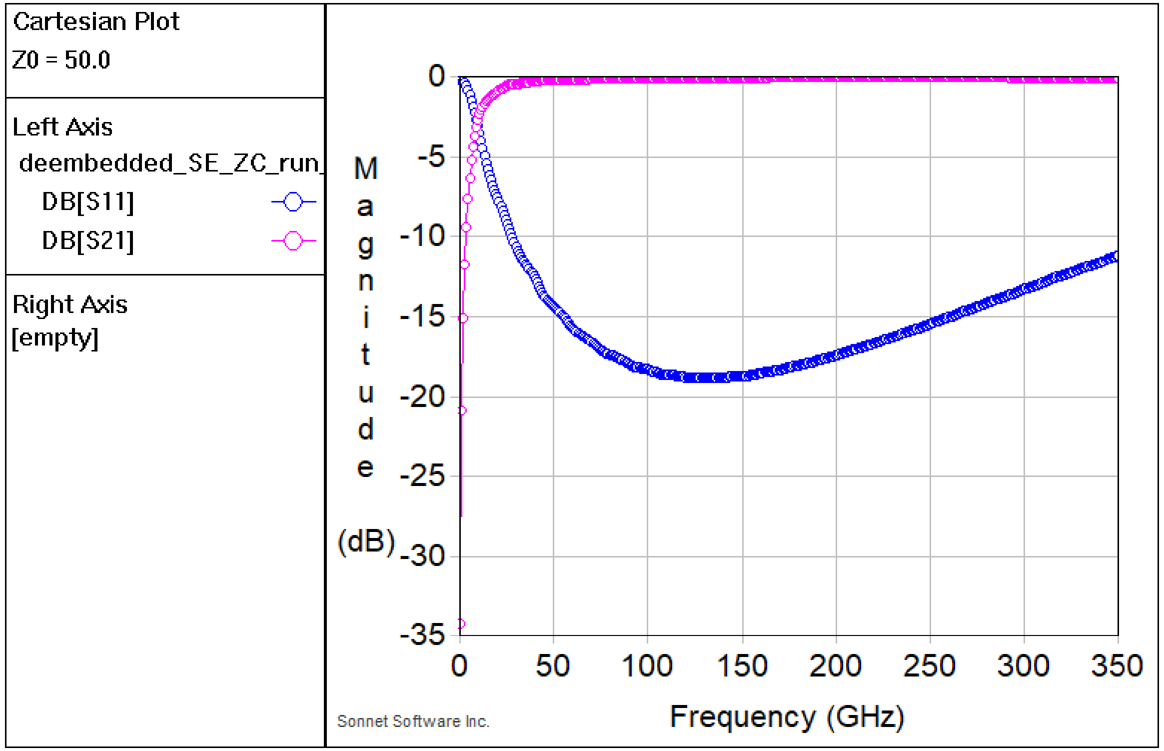
- Simulate DUT with 25µm feed line added on each side
- Simulate 2 x 25µm thru line
- De-embeded DUT using thru line data (Amphenol software)



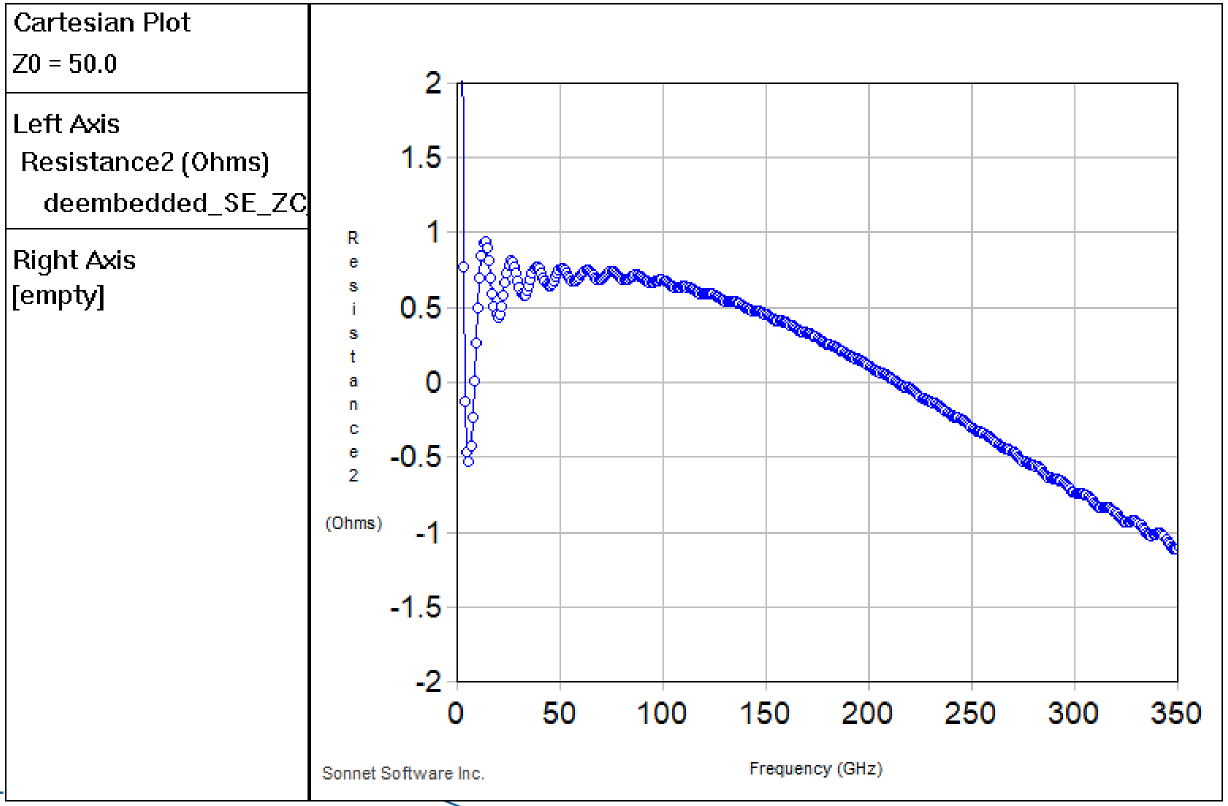
Reference simulation in Sonnet EM (MoM)



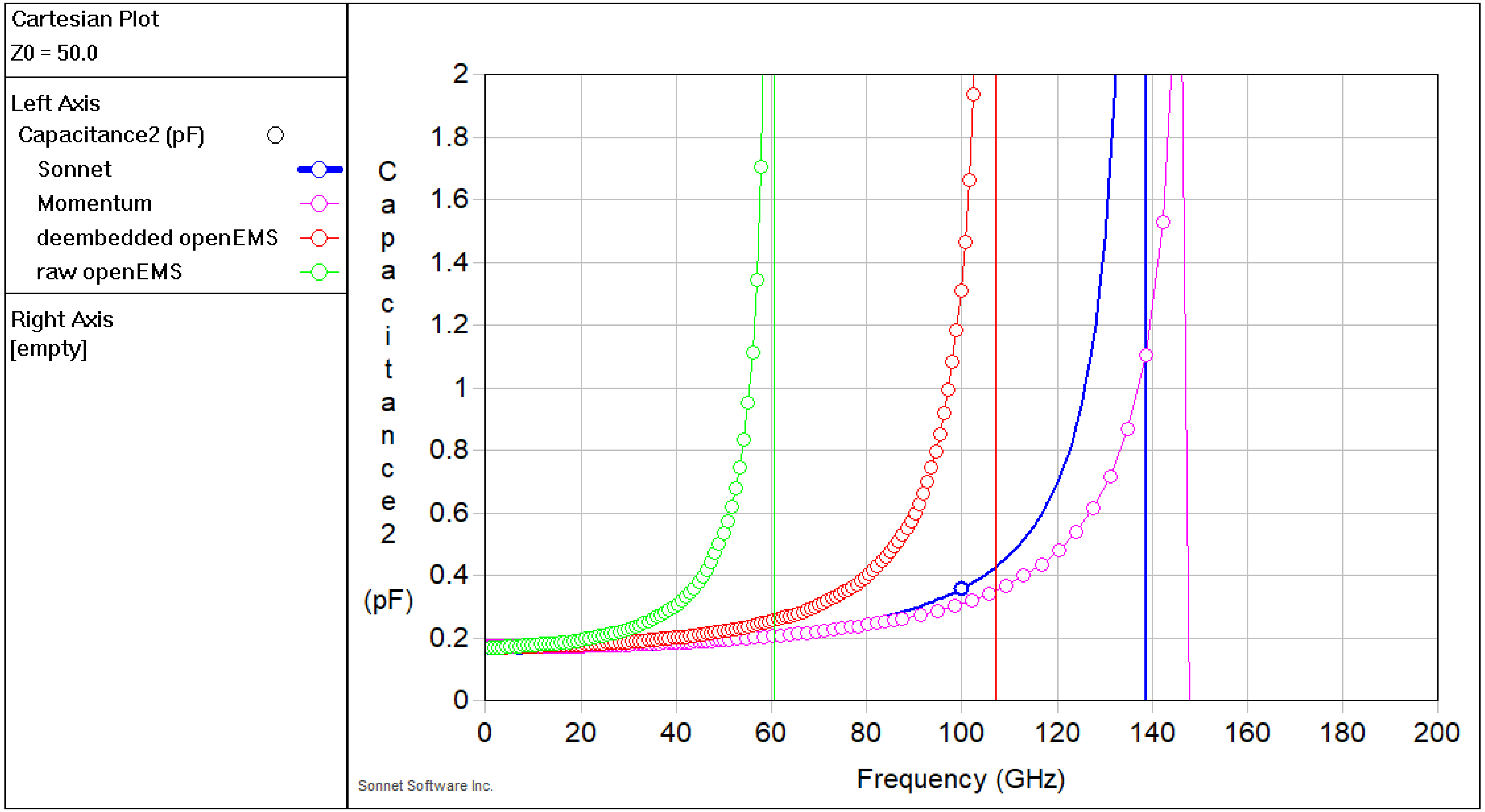
De-embedded, removing ports + 25µm each



- Data is passive.
  Negative series R looks strange, but no issues.



Series capacitance after de-embedding



Remove feedline, compare to MoM results

openEMS MIM de-embedded

Cseries = 171.4 fF

Lseries = 13.4 pH

Rseries = 0.69 Ohm

Cshunt1 = 3.46 fF (per side)

Momentum

Cseries = 171.3 fF

Lseries = 6.8 pH

Rseries = 1.12 Ohm

Cshunt = 4.92 fF

Sonnet

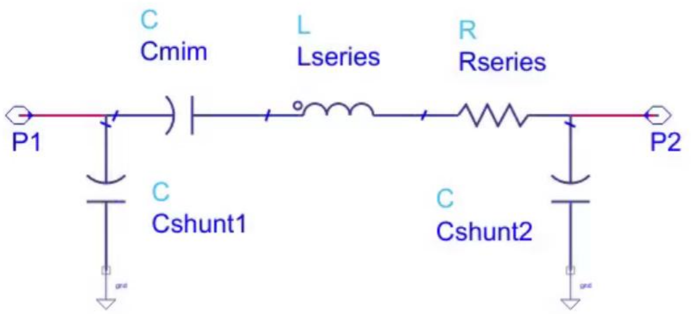
Cseries = 168.7 fF

Lseries = 8.1 pH

Rseries = 0.81 Ohm

Cshunt = 4.65 fF

values extracted at 70 GHz



Summary

- Port de-embedding required for DUT where port size is on the order of DUT size
- Simply speaking, lumped ports add a few pH of series L
- After de-embedding, MIM SRF is still lower than Sonnet and Momentum