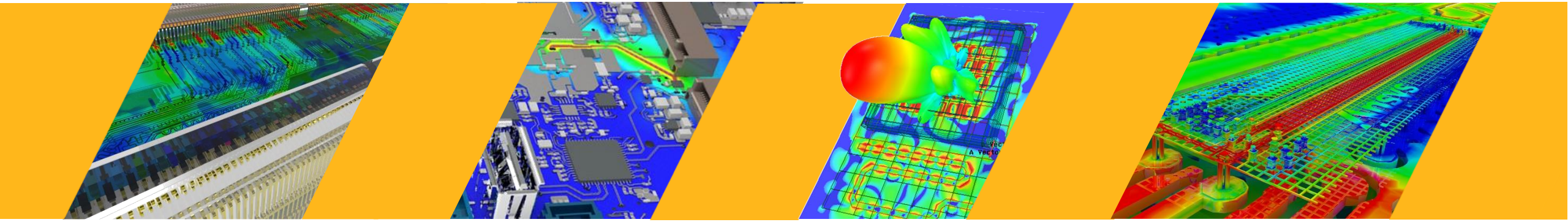
Powering Innovation That Drives Human Advancement

2025 Ansys Updates

19th March 2025

Brandon Kim
Senior Principal Application Engineer

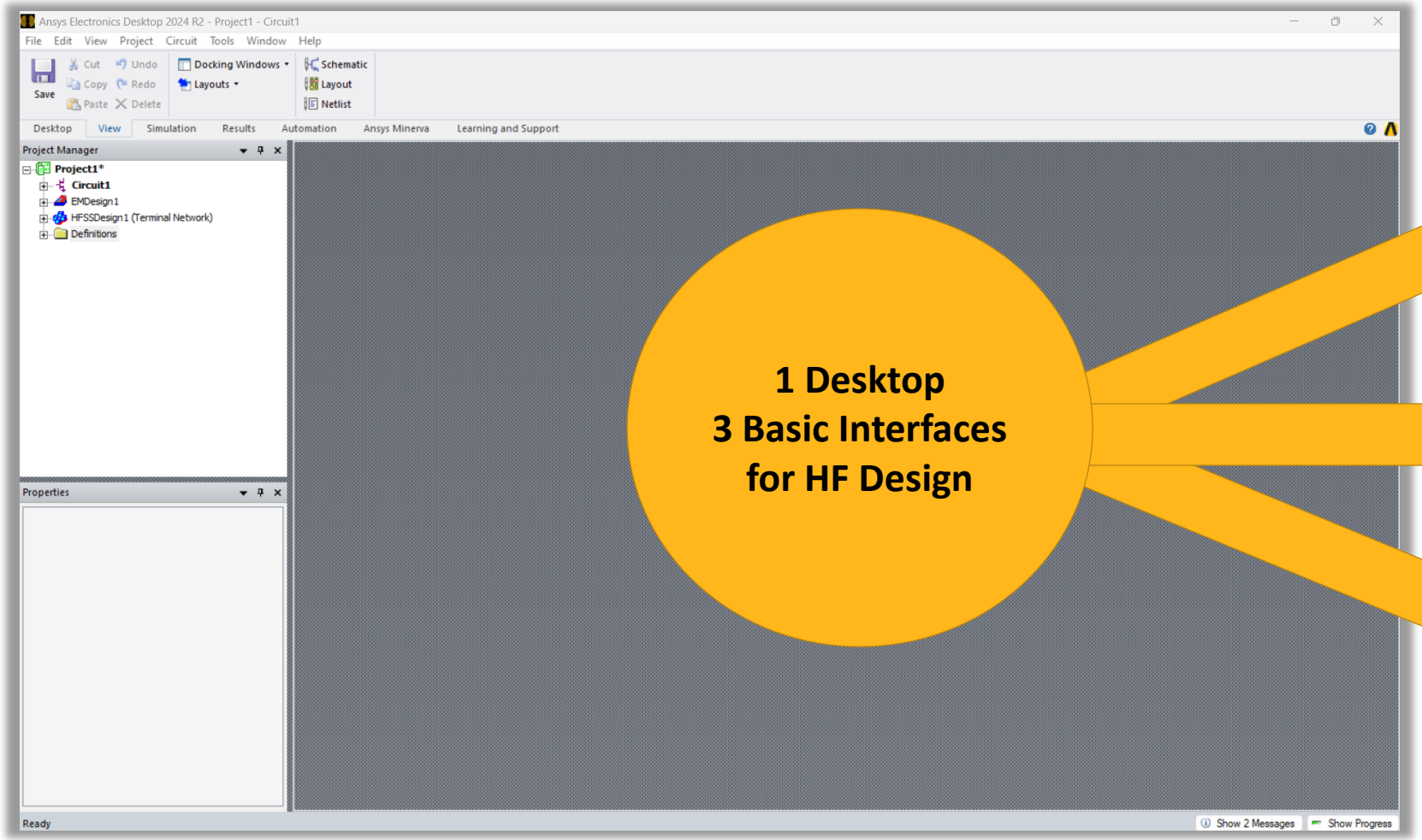
Agenda

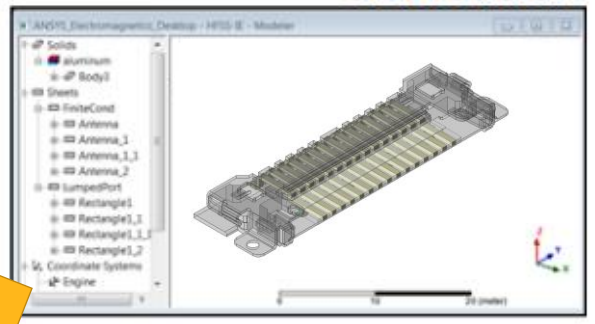


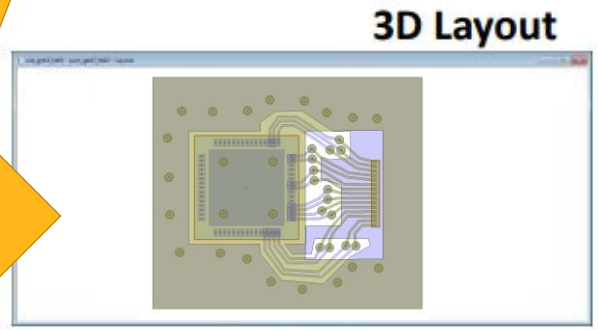
1. 강화된 Layout Component 를 이용한 System Assembly
2. Q3D in HFSS 3D Layout
3. Optislang – Metamodel Automation
4. ADRP (AI-Driven Routing Project)

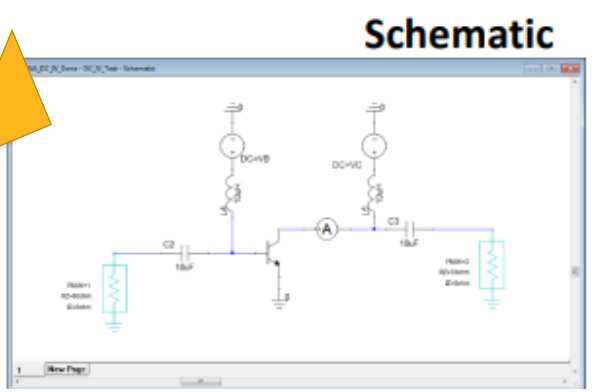
1. 강화된 Layout Component 를 이용한 System Assembly

AEDT (Ansys Electronics Desktop) Platform





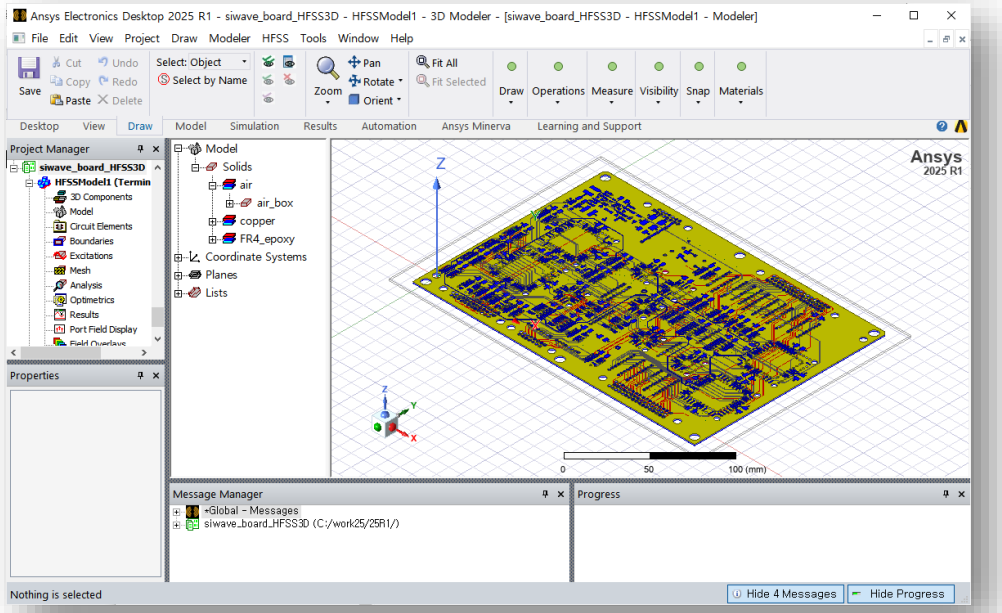




HFSS, HFSS 3D Layout, Circuit 을 하나의 AEDT 환경에서 작업

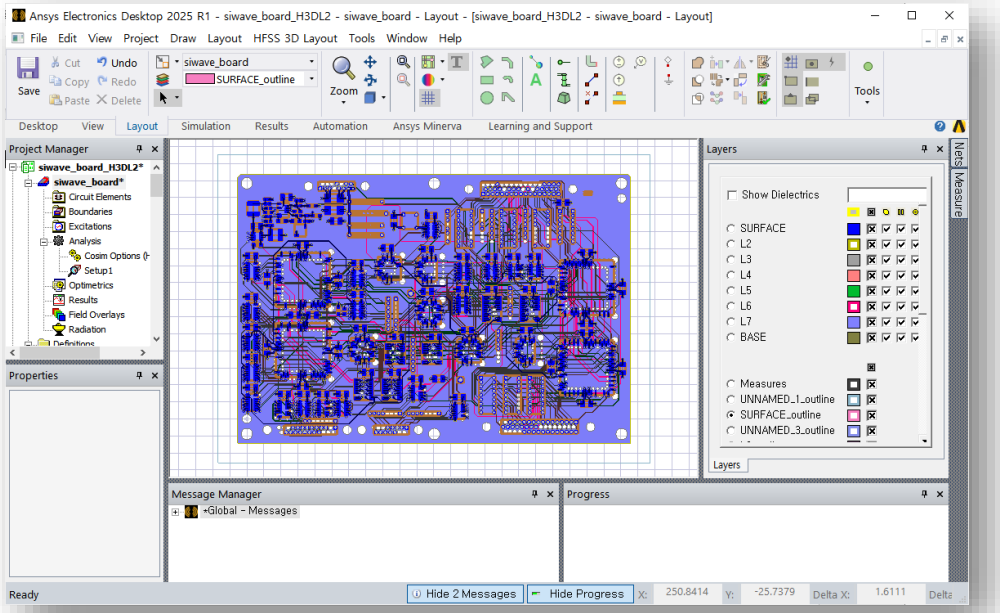
HFSS vs. HFSS 3D Layout

HFSS (115MB)



- Full 3D 해석용 Interface
- PCB 속성 없이 구조적 형상과 재질만 존재
- 상대적으로 무거운 3D GUI
- **해석 형상에 제한이 없음**
- 40년 역사의 3D 해석 golden standard

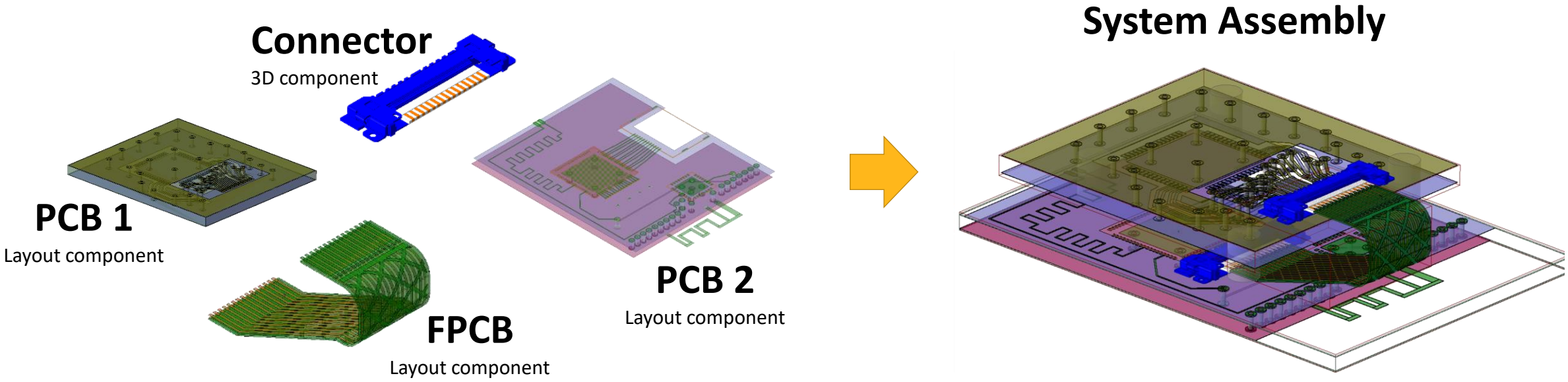
HFSS 3D Layout (5MB)



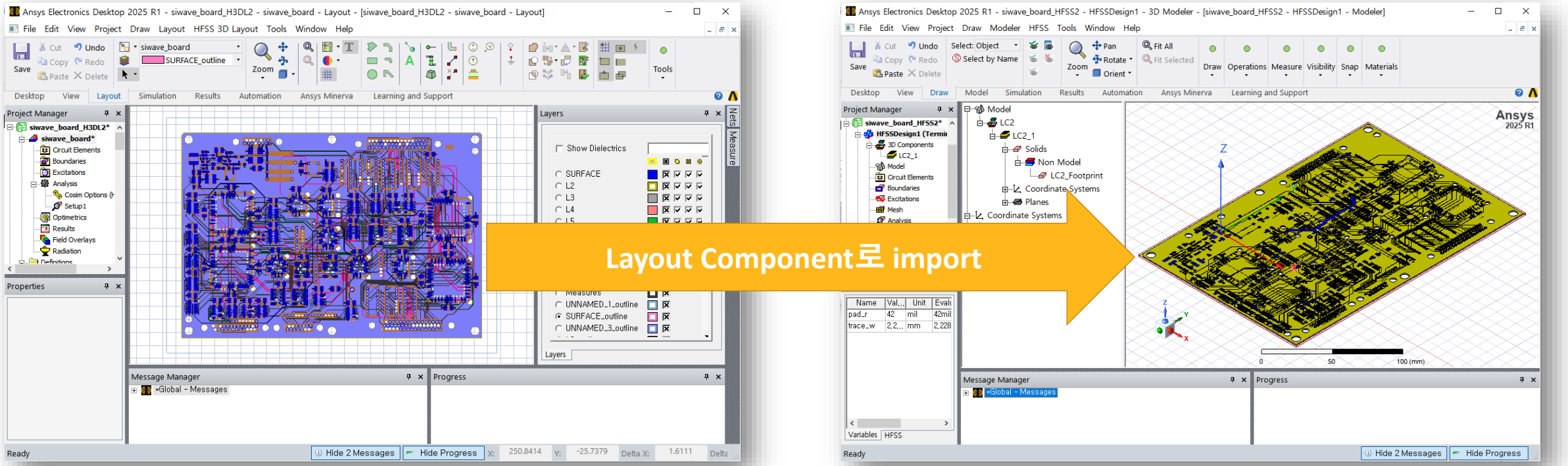
- 적층 구조의 **PCB 해석에 최적화**된 Interface
- net/padstack/component/stackup 속성 관리 가능
- 경량의 GUI로 크고 복잡한 PCB도 빠르게 작업 가능
- PCB에 최적화된 mesh 환경으로 평균 x2배 속도 향상
- 고속의 SIwave 및 Q3D solver도 해석 가능

NEW 3D System Assembly – Layout Component + 3D component

- Layout Component 적용시 Phi Plus 및 TAU가 Mesh Fusion 사용시 적용 가능
- Layout Component 기능 업그레이드 - snapping
- Mesh Fusion을 위한 Mesh Feedback 향상
- Component 교차(Intersection) 처리 및 우선 순위 적용



What is Layout Component ?



- 3D component를 import 하듯이, HFSS 3D Layout의 가벼운 PCB 데이터를 HFSS로 import하는 기능
- 이를 통해 **HFSS 3D Layout의 가벼운 GUI** 와 **모든 형상을 다룰 수 있는 HFSS의 장점**을 모두 활용 가능

강화된 25R1 Layout Component

1. Real time GUI Interface

- 기존 HFSS 환경에서 HFSS 3D Layout 과 동일한 속도로 실시간 회전/확대가 가능한 고속 GUI
- **가볍고 빨라진 GUI**로 작업효율을 크게 향상 (port설정,수정,3D 구조 추가..)

2. PCB 속성들을 HFSS 에서도 변수로 수정 가능 25R1

- trace 폭/via 크기/pad 크기와 같은 **PCB 파라미터를 변수화** 가능
- 기존 full 3D 환경은 외곽선 좌표로만 객체가 구성되어 수정이 어려움

3. Layout component 내의 구조물에 대한 snap 가능 25R1

- 3D component 추가 시 PCB 구조물의 모서리/변 등을 마우스로 **snap**하여 작업 가능

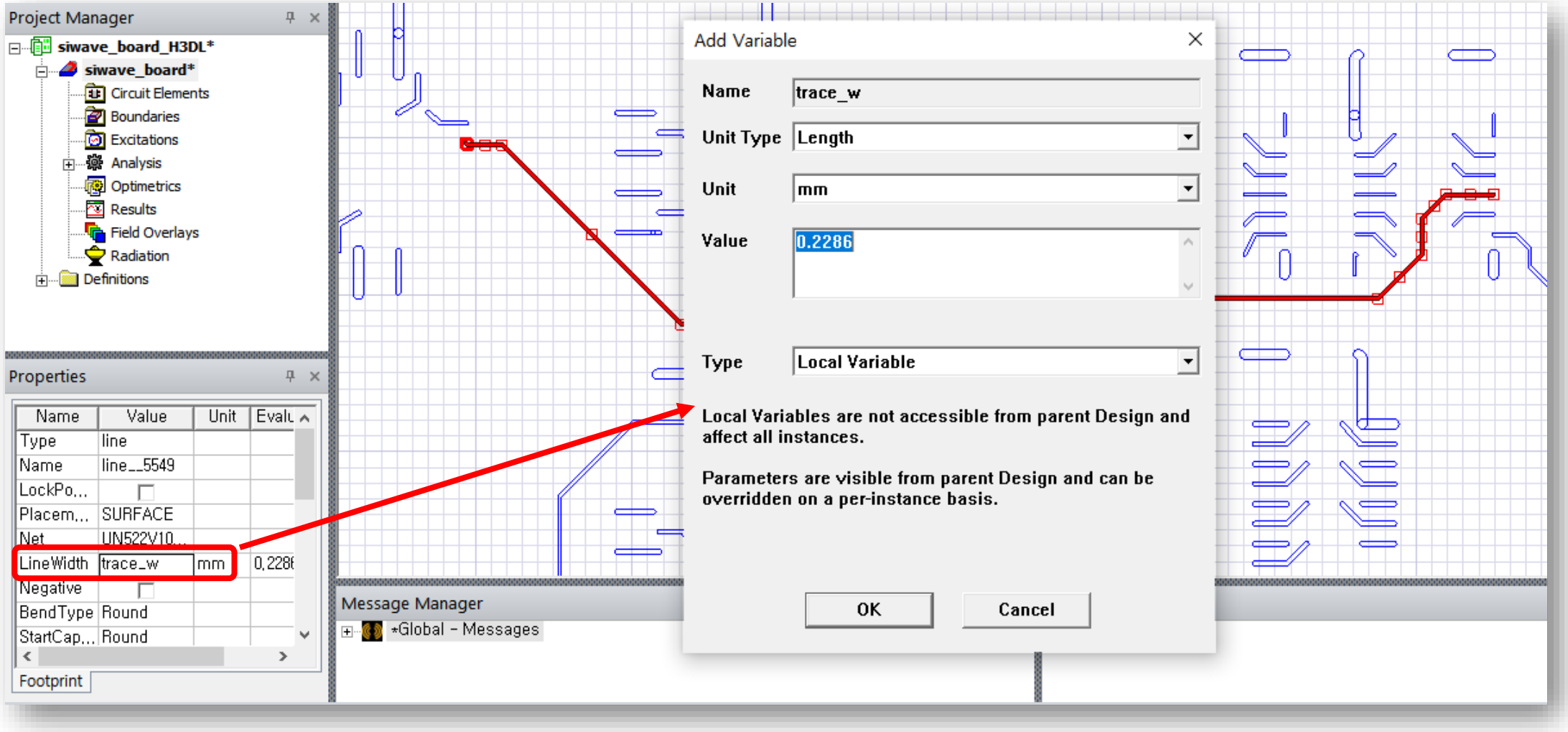
4. 다른 3D component와 함께 해석 시 모든 종류의 Mesh Fusion 가능 25R1

- Phi plus와 tau를 비롯한 **모든 종류의 initial mesh** 를 각자의 구조에 맞게 개별 적용 가능

5. 3D component 와의 겹치는 구간 개선 25R1

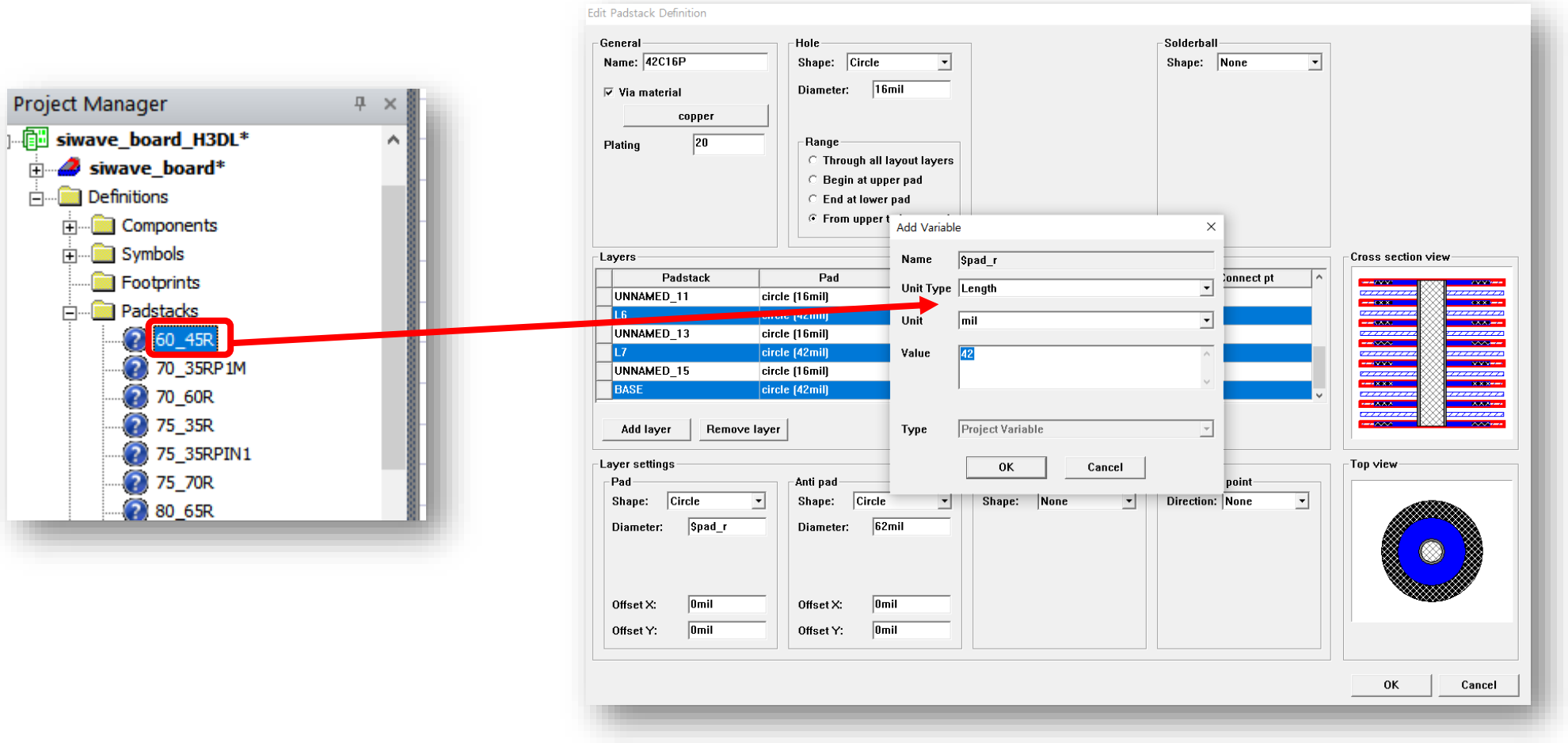
- Layout component 와 3D component 가 **겹치는 영역의 우선순위 설정 기능** 추가

HFSS 3D Layout : PCB 변수 설정 (예시 1 – 특정 trace 선폭)



- 마우스로 Trace 선택 후, Line Width 에 숫자대신 **trace_w** 문자 입력으로 변수화

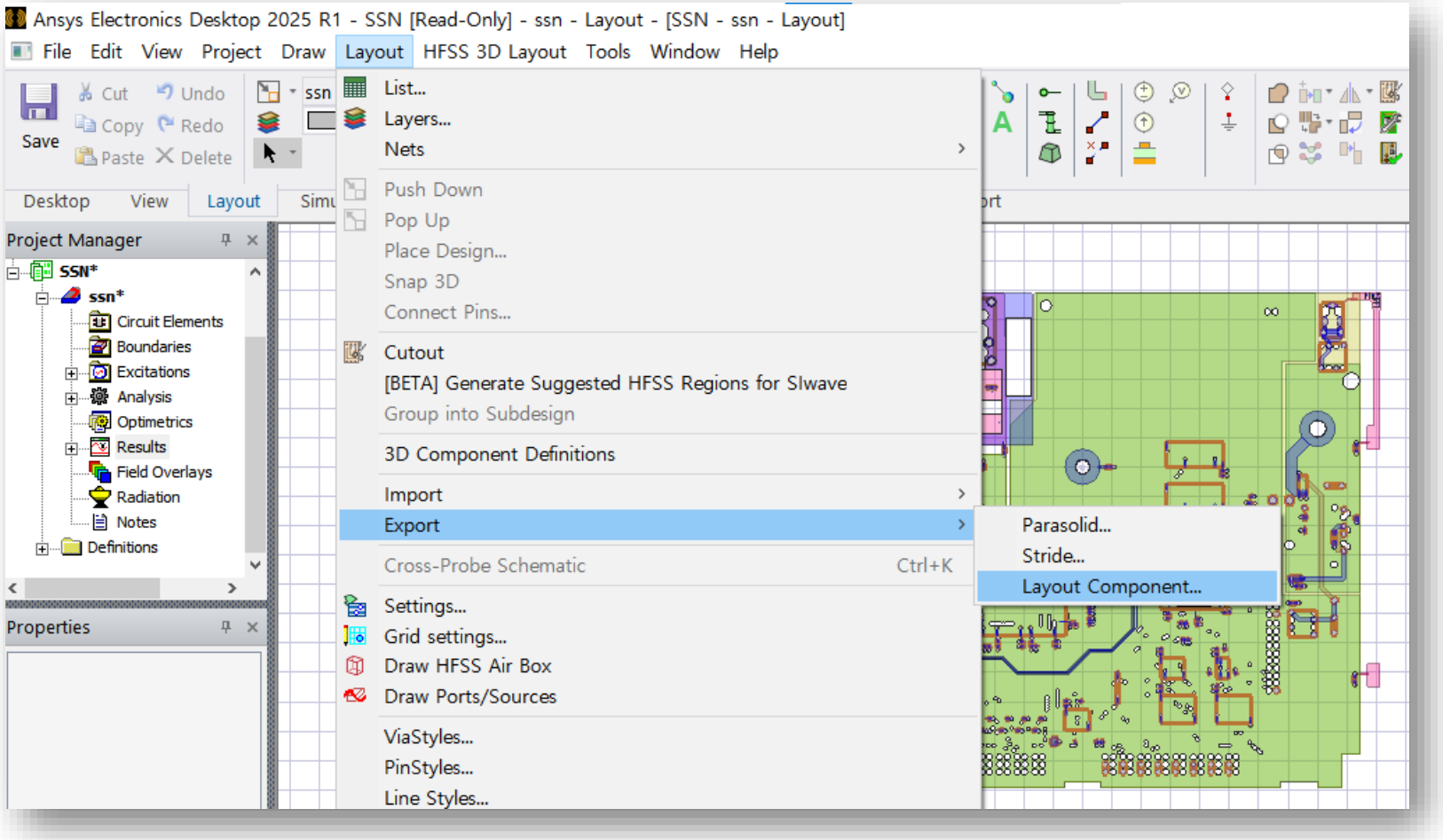
HFSS 3D Layout : PCB 변수 설정 (예시 2 – 특정 via pad 크기)



- Definition → Padstack에서 특정 padstack을 선택하고 모든 Pad의 Diameter에 **$pad_r** 입력으로 변수화
- Padstack의 경우는 공통 DB이기 때문에, 변수 앞에 "**$**"를 붙여서 project 전역변수로 설정해야 함.

HFSS 3D Layout : Layout Component 만들기

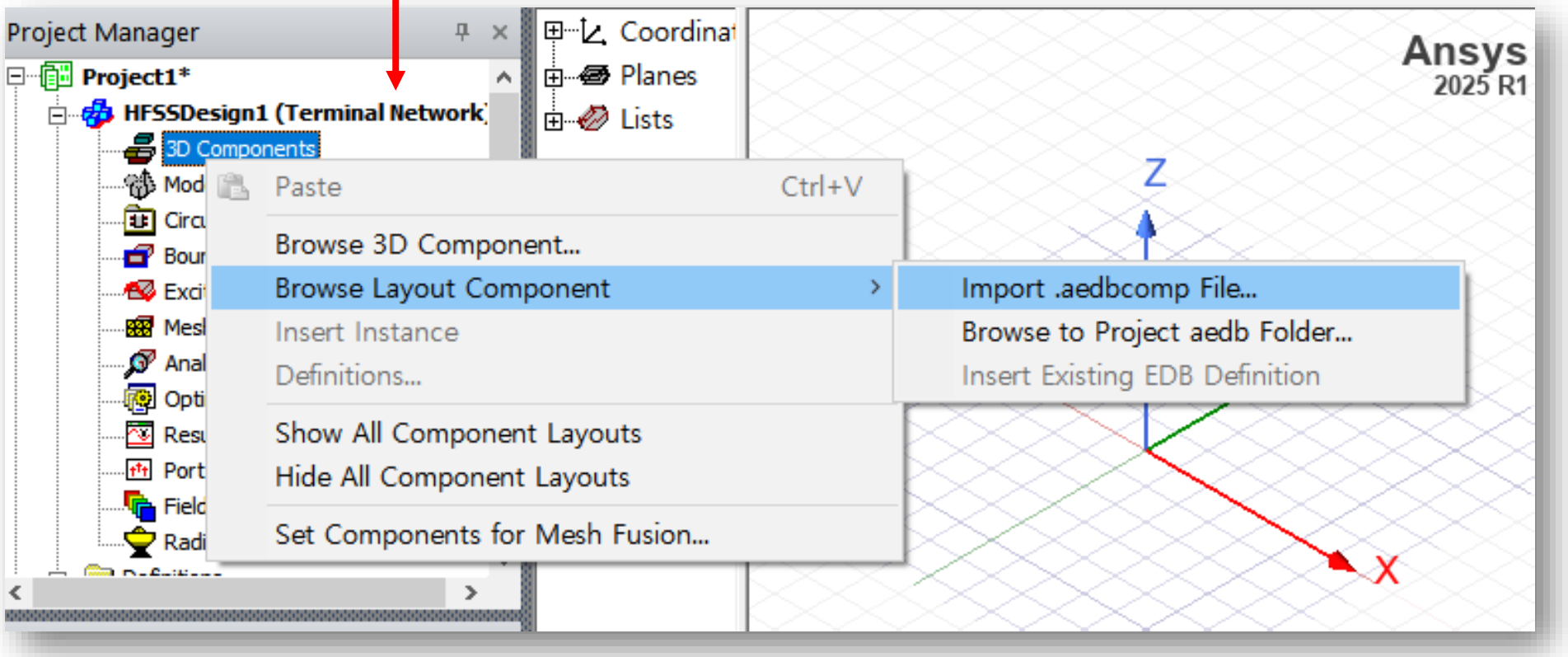
- 메뉴에서 Layout → Export → Layout Component



HFSS : Layout Component 가져오기

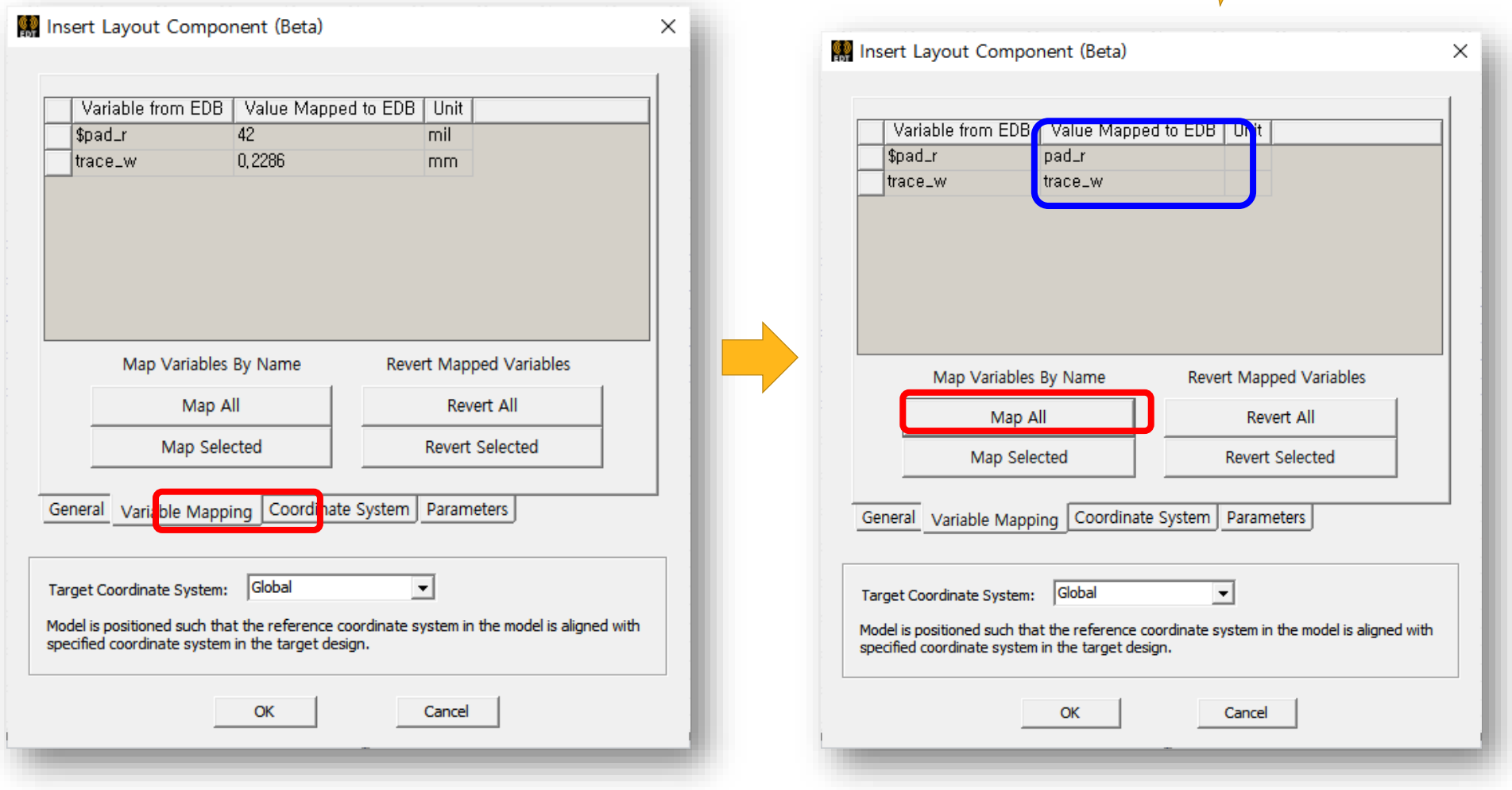
- HFSS 의 3D Components 탭에서 우클릭 → Browse Layout Component → Import .aedbcomp file 선택

Solution Type은 반드시 Terminal로 되어 있어야 함.



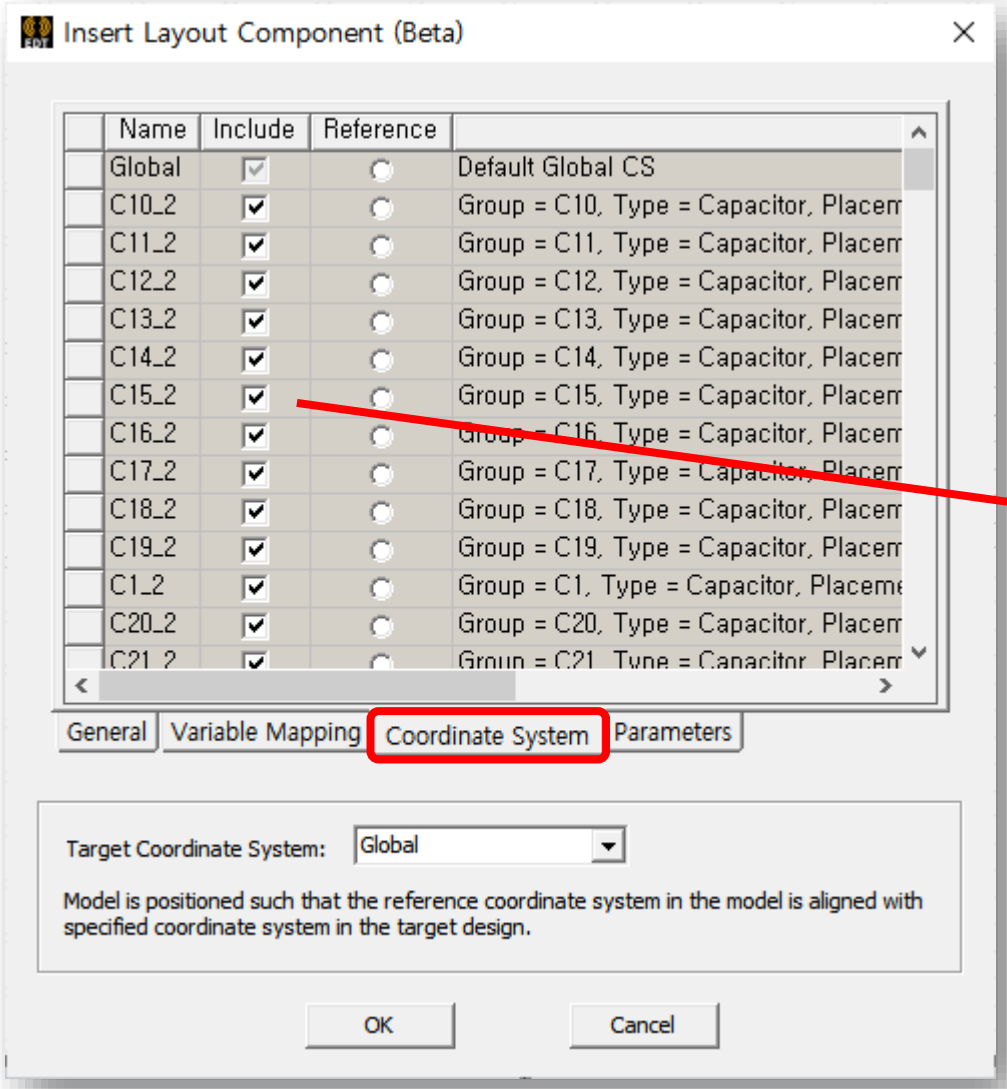
HFSS : Import Layout Component – 변수 매핑

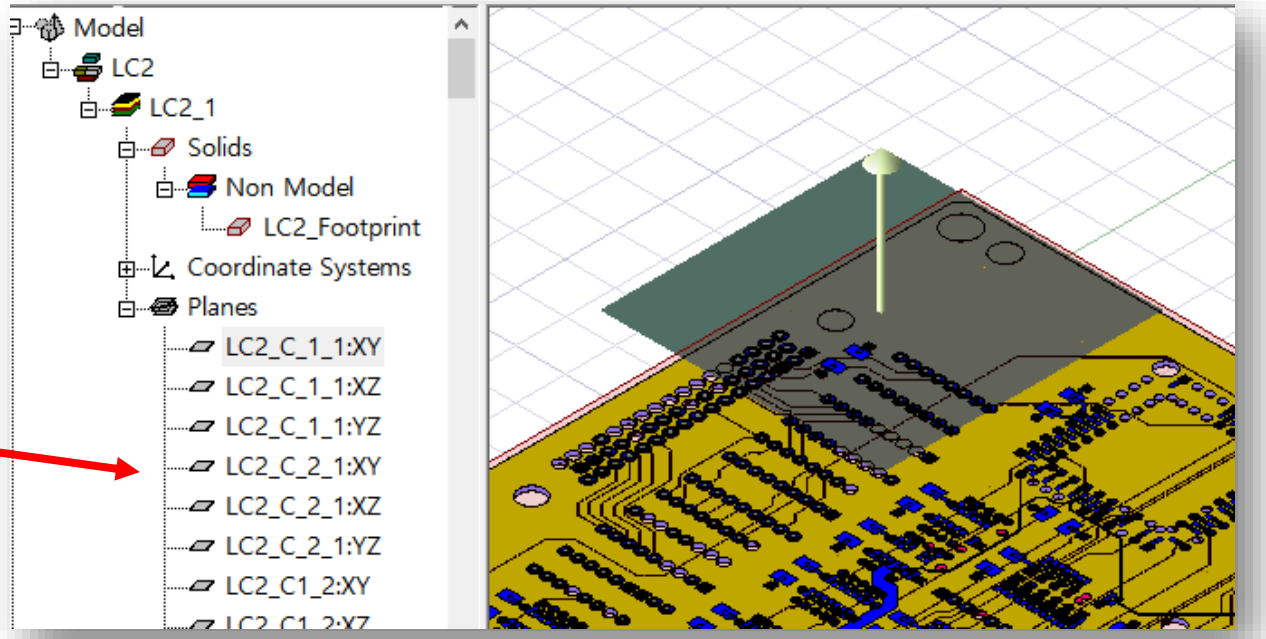
NEW



- Import 창에서 Variable Mapping 탭 → Map All 을 클릭하면 HFSS 3D Layout에 설정한 변수들이 HFSS로 넘어옴

HFSS : Import Layout Component – 자동 상대좌표 추가

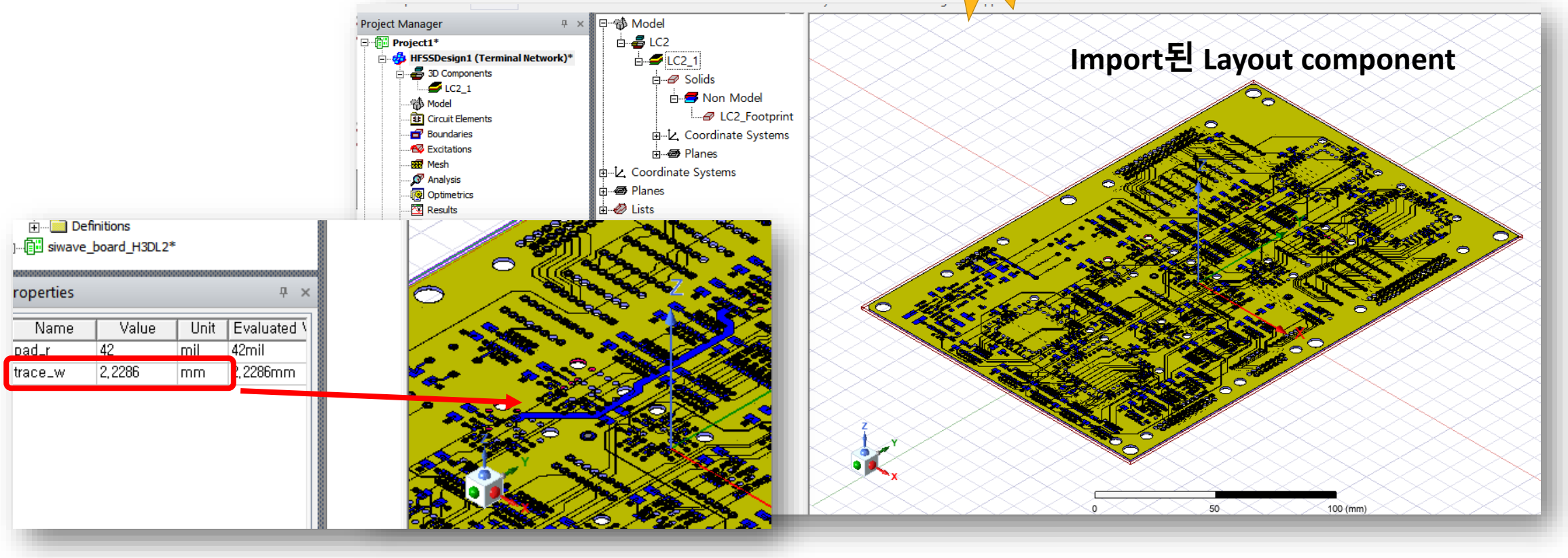




- Import 창에서 Coordinate System 탭 → 특정 소자 pad를 선택하면 해당 pad 위치에 자동으로 가상좌표(CS)를 만들어줌
- 이 가상좌표를 이용하여 3D component를 추가할 때 간편하게 위치를 지정할 수 있음

HFSS : Layout component의 변수 조절

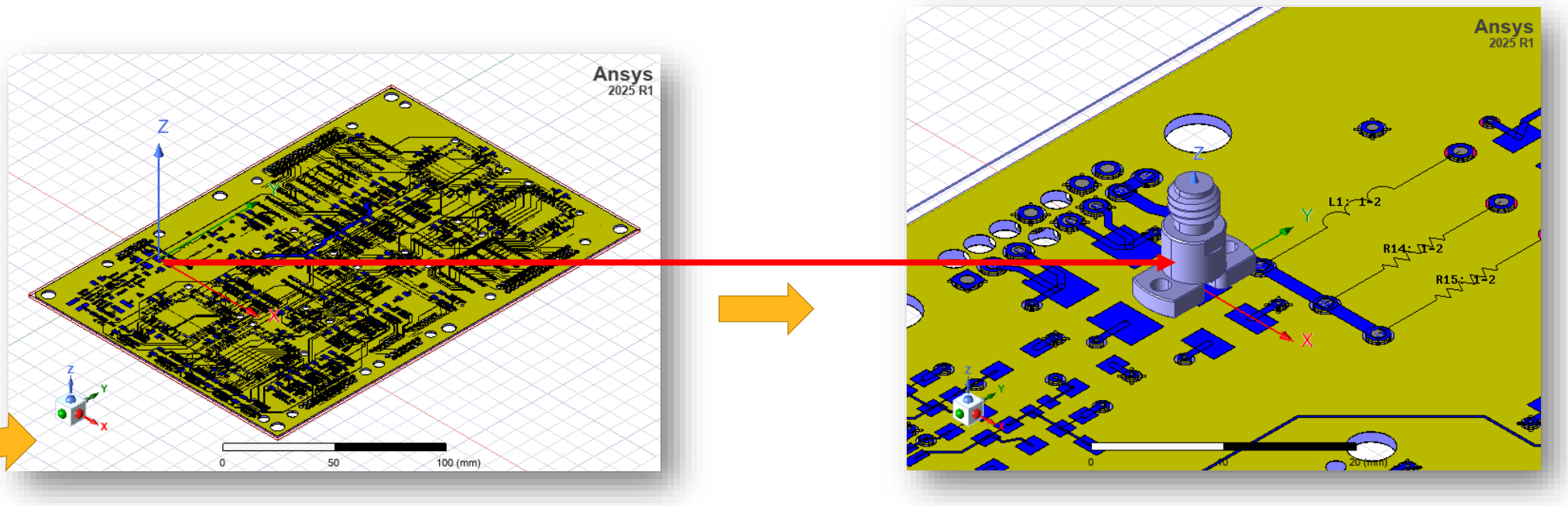
NEW

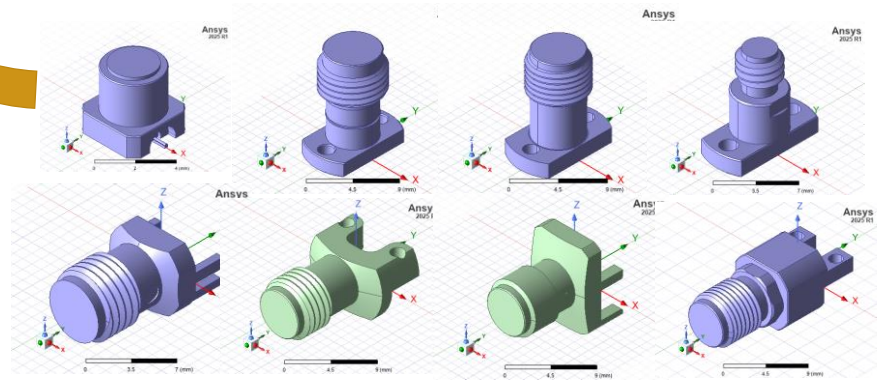


- HFSS 3D Layout에서 넘어온 trace_w 변수를 수정하면, Layout component의 해당 선폭이 실시간으로 가변됨

→ 이를 이용하여 HFSS 에서 선폭/Pad크기/via 좌표 등등 모든 종류의 PCB CAD 데이터를 변수화 가능
→ 여기에 HFSS 3D Layout처럼 가벼워진 GUI는 덤!

3D Component 추가하기





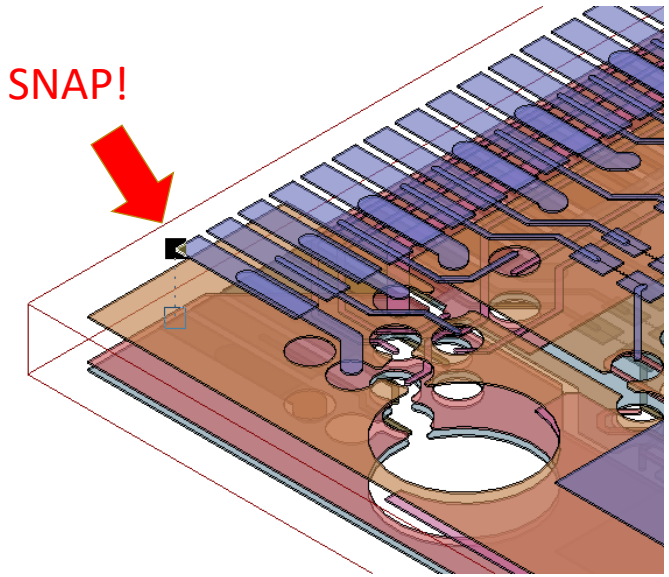
Connector 3D component models : withwave™

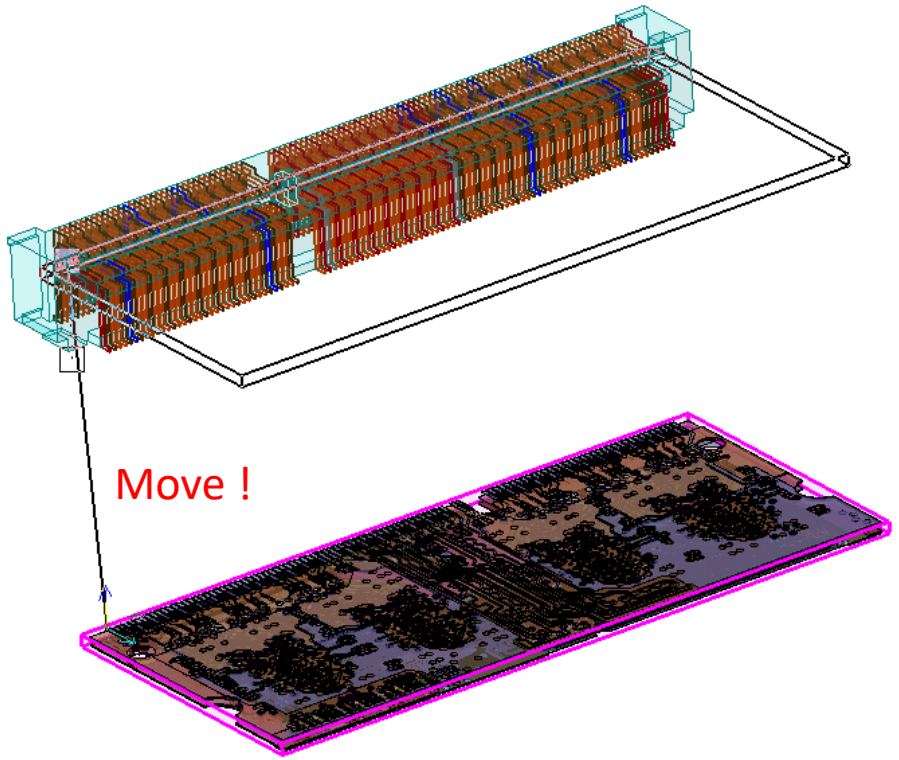
- 제조사로부터 Connector, Cable, module, PKG 등과 같은 HFSS 3D component 파일을 수령 (.a3dcomp)
- 윈도우 탐색기에서 .a3dcomp 파일을 HFSS 화면으로 Drag & drop하여 간단하게 추가
- 3D component가 추가될 위치는 HFSS의 좌표 원점.
  상대좌표(CS)를 선택 또는 생성하여 임의 위치에 배치 가능

Snap to Layout Component Geometry

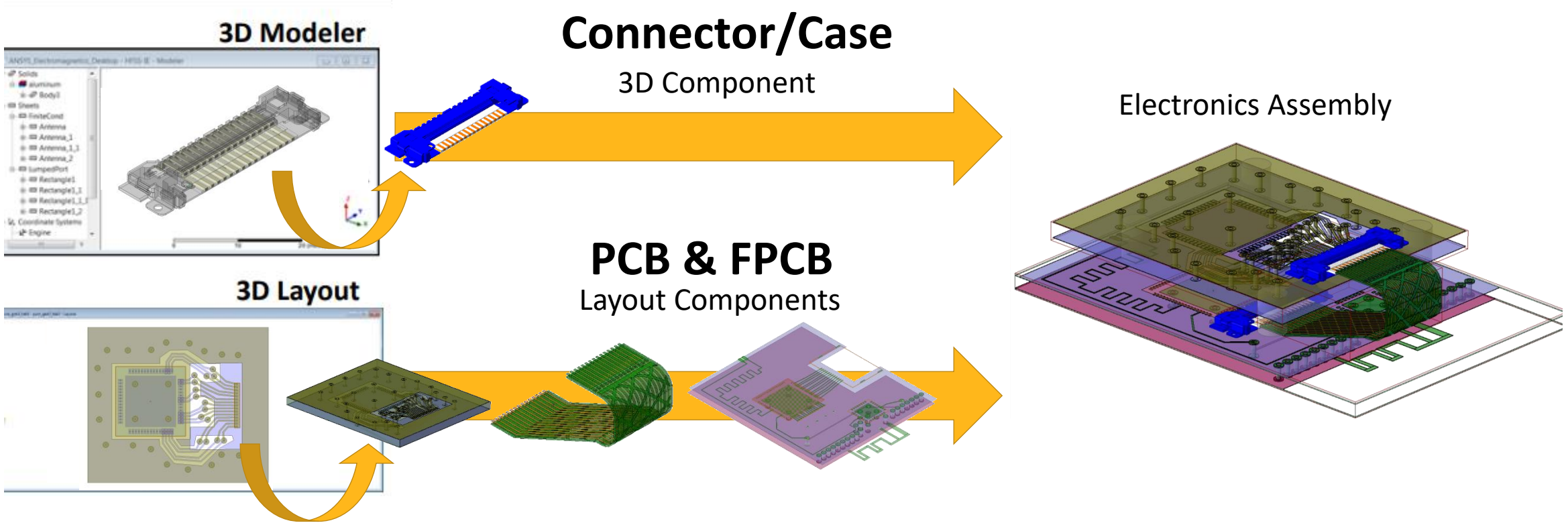
NEW

- HFSS로 import 된 Layout Component의 PCB 모든 위치를 마우스로 snap 가능
  - vertex, face center, edge center, edge quadrant, arc center
- PCB와 Connector 간의 pin-to-pin 정렬과 Component 배치가 훨씬 용이



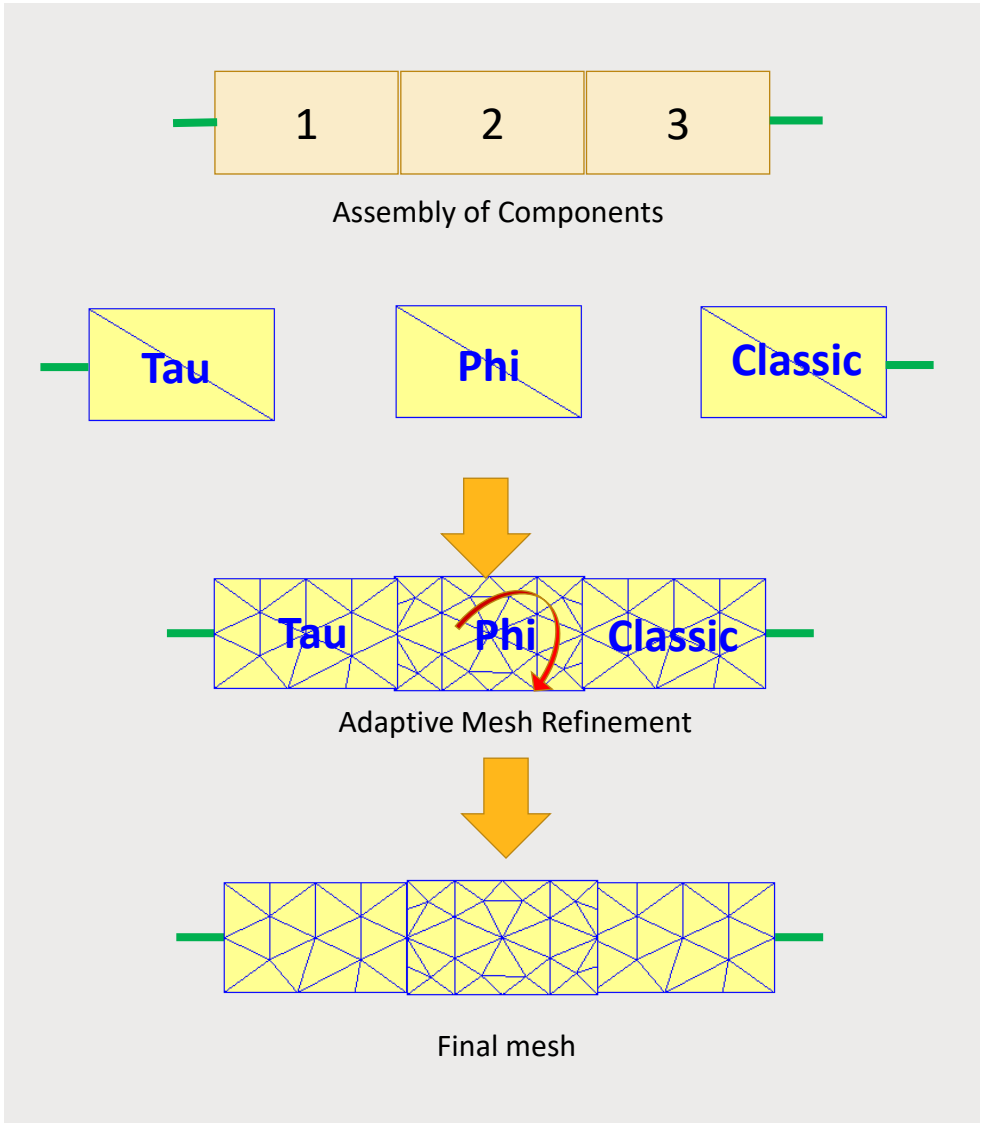


3D System Assembly in HFSS

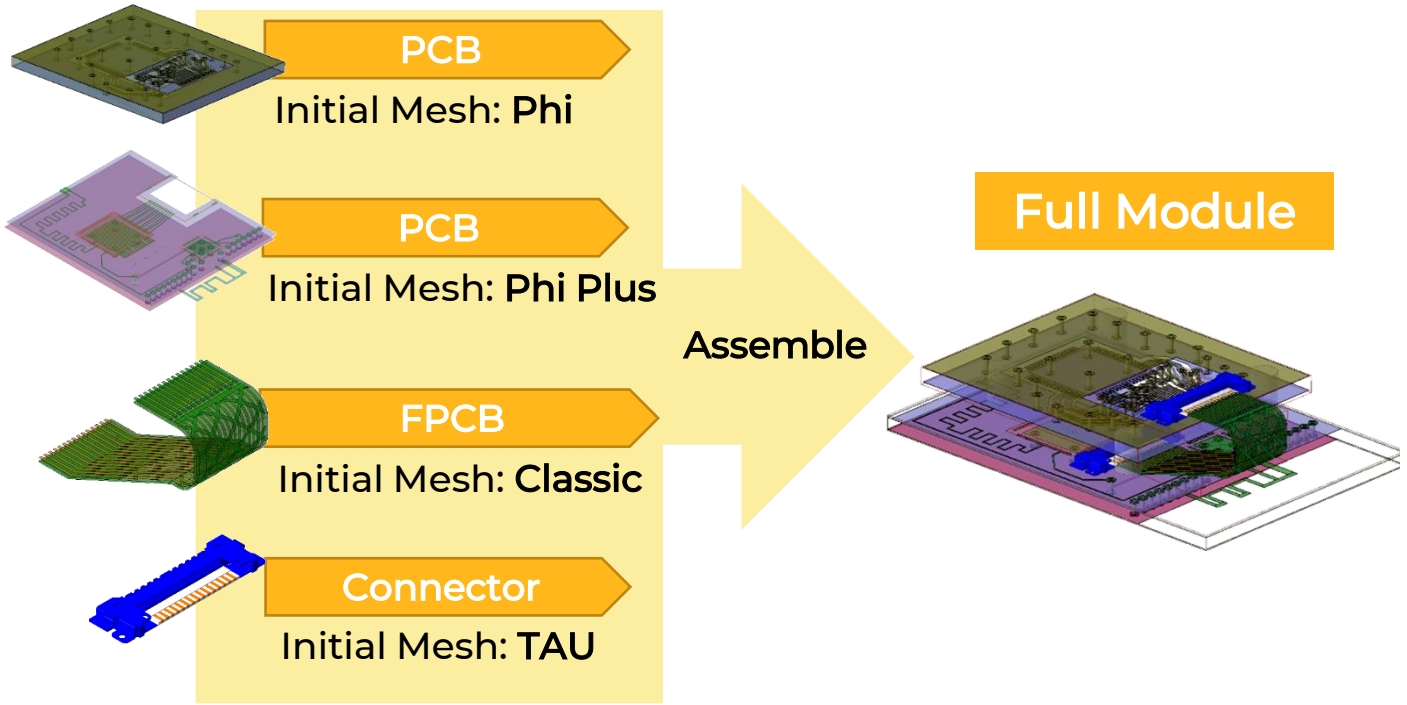


- 기존의 HFSS MCAD 3D 에서 HFSS 3D Layout의 PCB 모델을 Layout Component로 import
- 여기에 full 3D 형태의 connector를 함께 attach하여 효율적인 3D system을 구축
- 결과적으로 기존의 **HFSS와 HFSS 3D Layout의 장점을 모두 활용 가능** (모델링 제한 X, 빠른 GUI, Mesh Fusion)

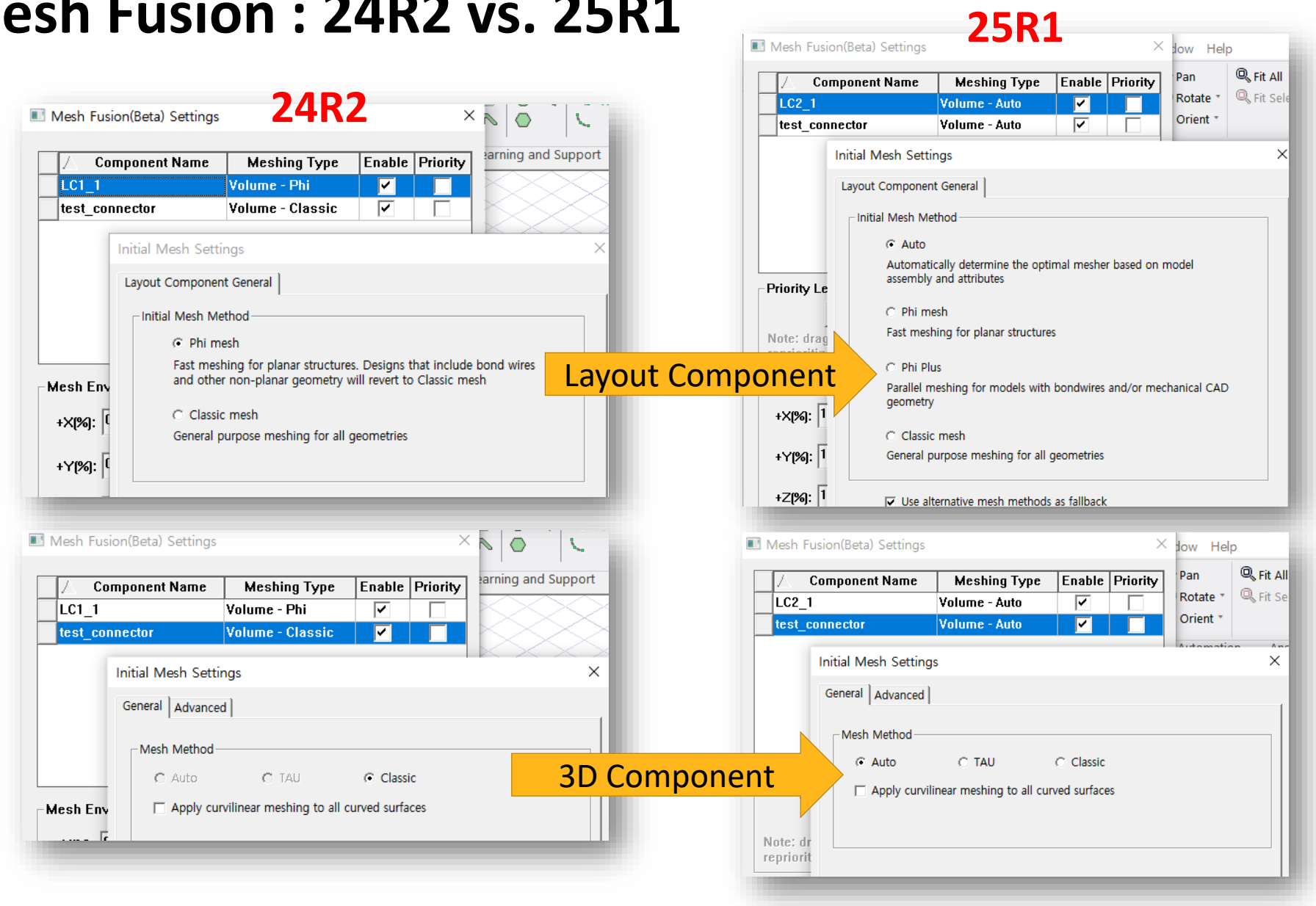
Mesh Fusion – 모든 mesh 조합 가능



각 3D component와 Layout component 별로 최적의 initial mesh 선택 → 모든 종류의 mesh를 선택 가능 25R1



Mesh Fusion : 24R2 vs. 25R1

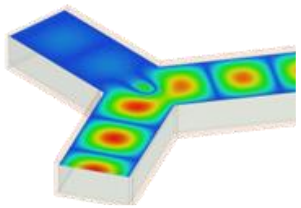


NEW

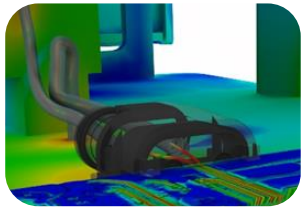
- 25R1 부터는
  모든 종류의 mesh를
  Mesh fusion에 적용 가능
- 각 component 형상별
  최적의 mesh를 찾아주는
  auto 기능도 활성화

HFSS Initial Meshing Algorithm의 이해

각 component의 구조에 적합한 Initial mesh

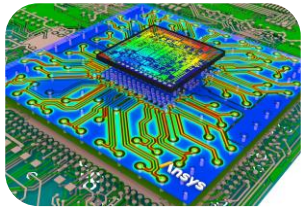


Classic

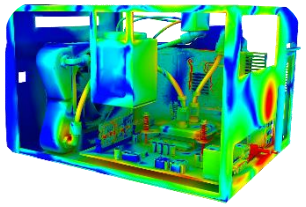


TAU

- **Classic** : HFSS의 가장 기본적인 mesh, 모든 구조의 3D 구조에 적용가능하다는 장점이 있지만, 가장 복잡하고 느림
- **TAU** : 원형 구조가 많은 복잡한 구조에 최적

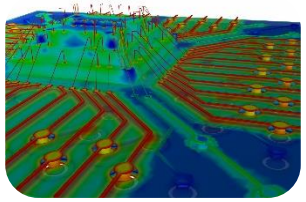


Phi

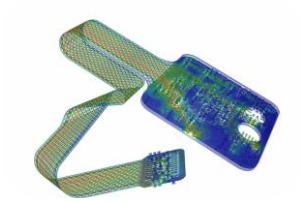


TAU Flex

- **Phi** : 2.5D PCB 적층 구조에 최적. 가장 빠른 mesh.
- **TAU Flex** : 오류가 많은 복잡한 3D 구조에 적용



Phi Plus

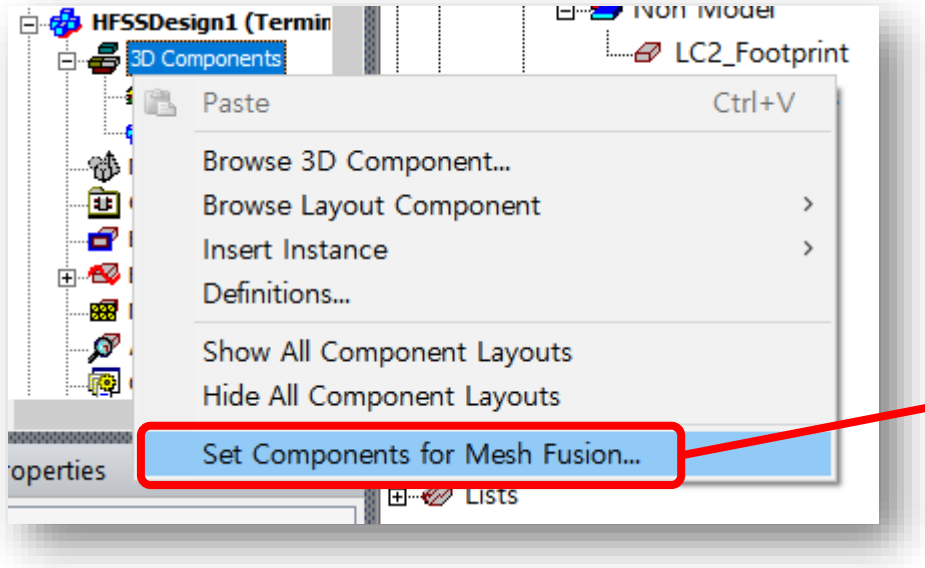


Classic Plus

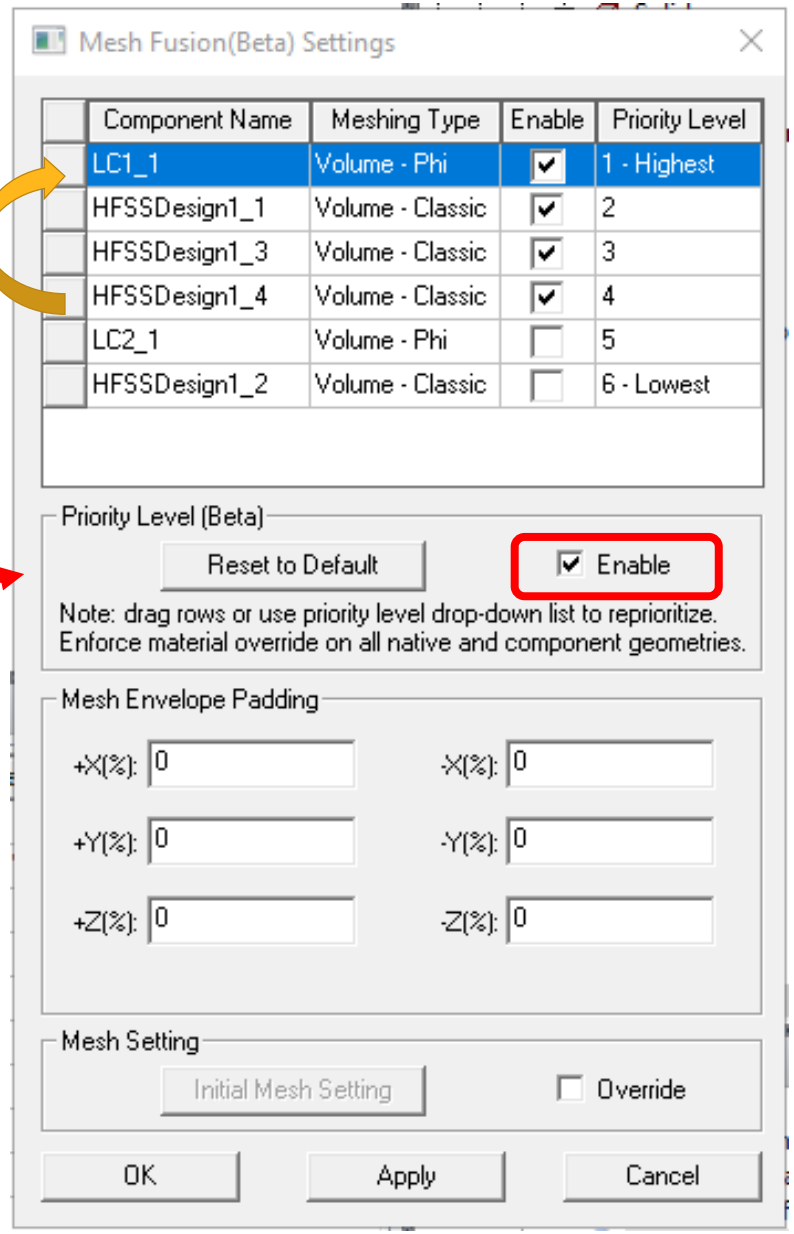
- **Phi Plus** : 본딩, 에칭과 같은 특수 3D 구조를 포함한 PCB 전용 mesh.
- **Classic Plus** : FPCB와 같이 휘어있는 PCB 구조에 적합

Component Priorities 25R1

NEW



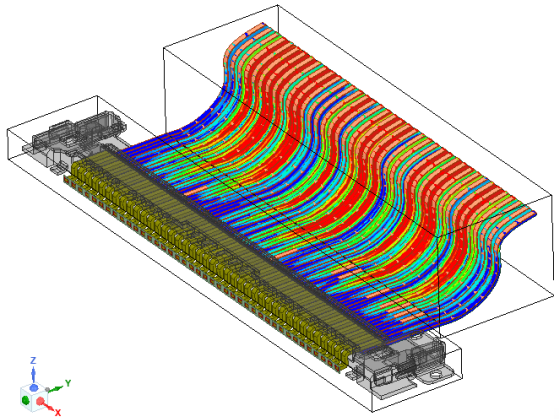
Drag & drop

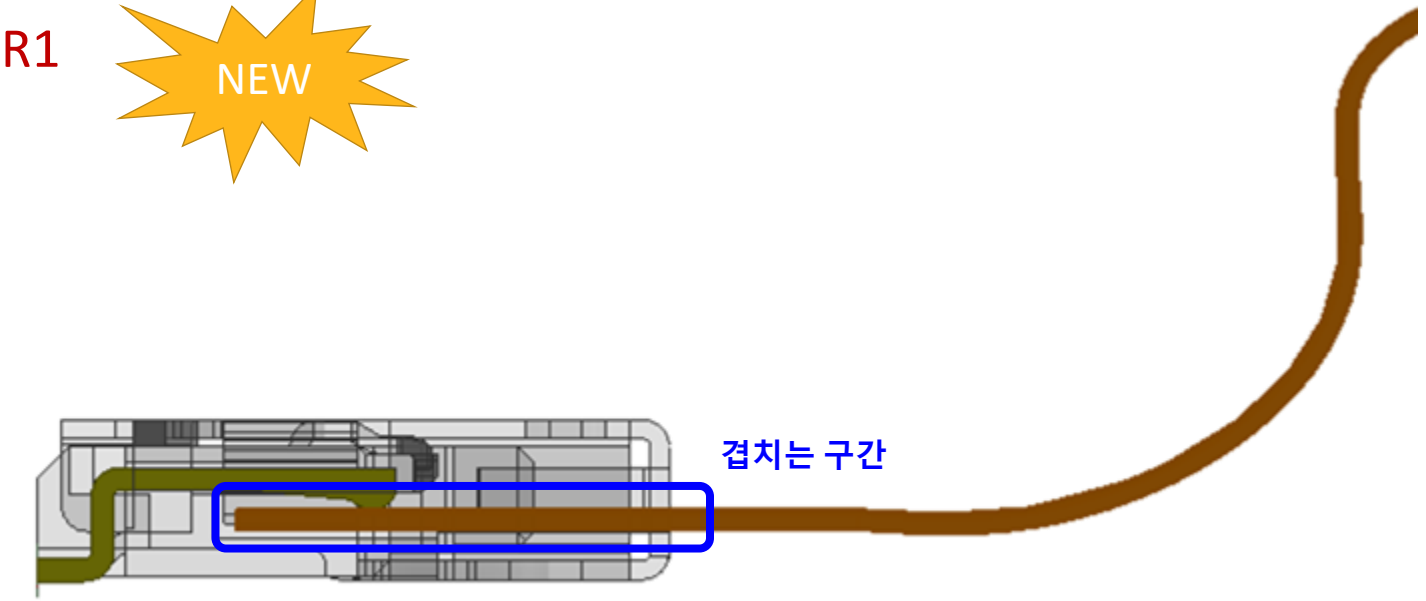


- Intersection을 해결하는 기존 규칙은 이상적이지 않을 수 있음
- 3D Components 에서 우클릭 → Set Components for Mesh fusion 에서 **각 component의 우선순위를 지정**할 수 있음
- 설정된 수준에 따라 component 끼리 교차될 때 자동으로 해결
- Drag & drop을 이용하여 우선 순위 변경 가능

Component Priorities[25R1]

NEW

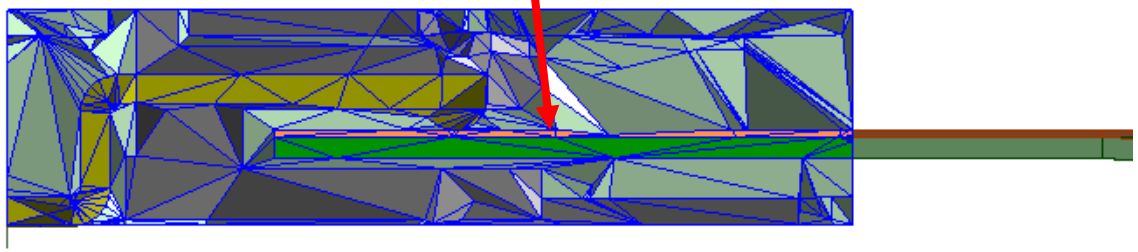




connector 에
우선권 할당

Mesh Fusion(Beta) Settings

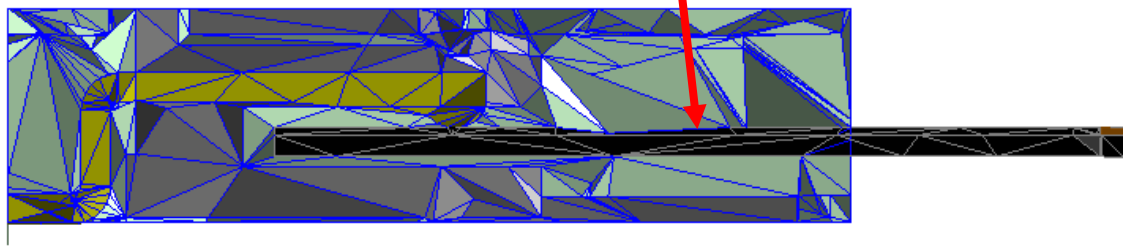
Component Name	Meshing Type	Enable	Priority Level
Ribbon_Cable_Connector	Volume - Auto	☑	1 - Highest
ribbon_cable2_1	Volume - Auto	☑	2
ribbon_cable1_1	Volume - Auto	☑	3 - Lowest



FPCB에
우선권 할당

Mesh Fusion(Beta) Settings

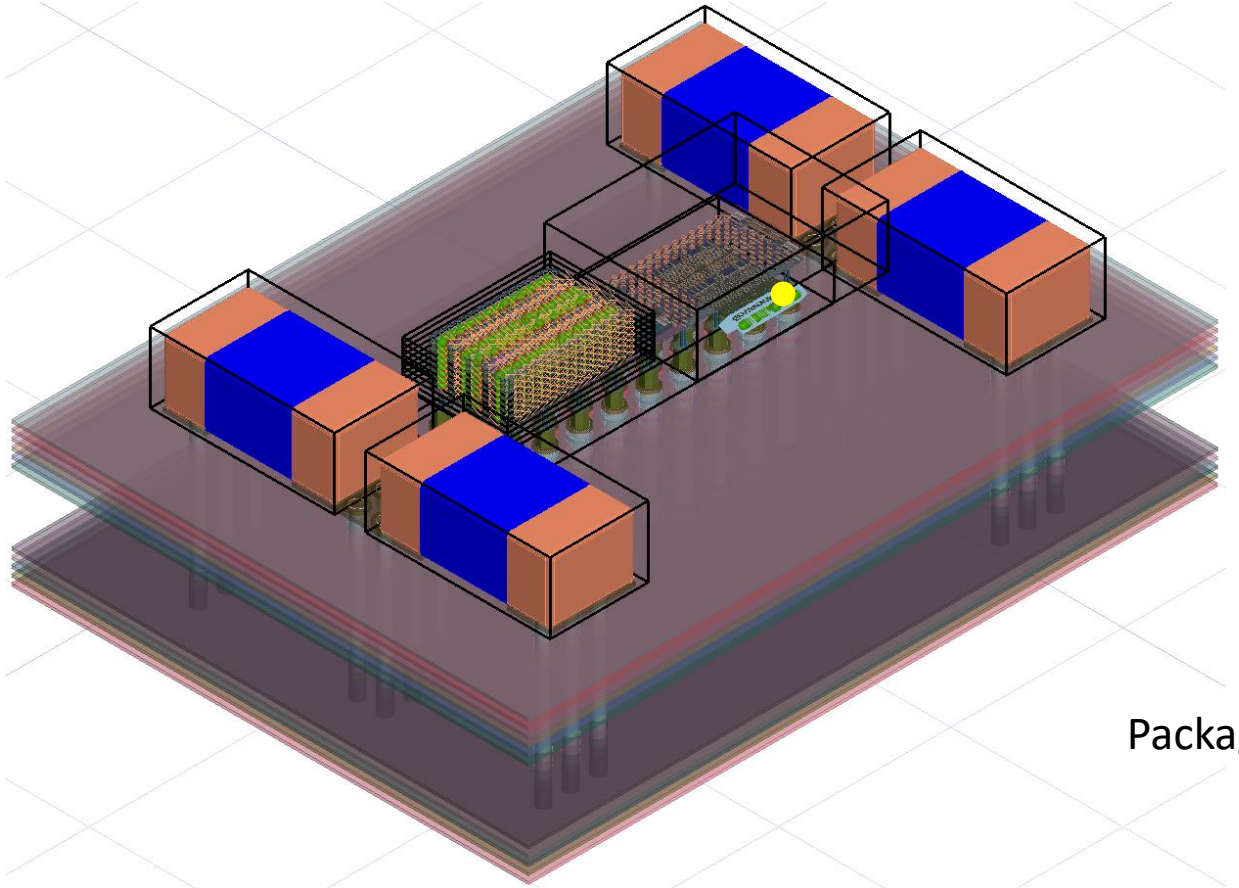
Component Name	Meshing Type	Enable	Priority Level
ribbon_cable2_1	Volume - Auto	☑	1 - Highest
ribbon_cable1_1	Volume - Auto	☑	2
Ribbon_Cable_Connector1	Volume - Auto	☑	3 - Lowest

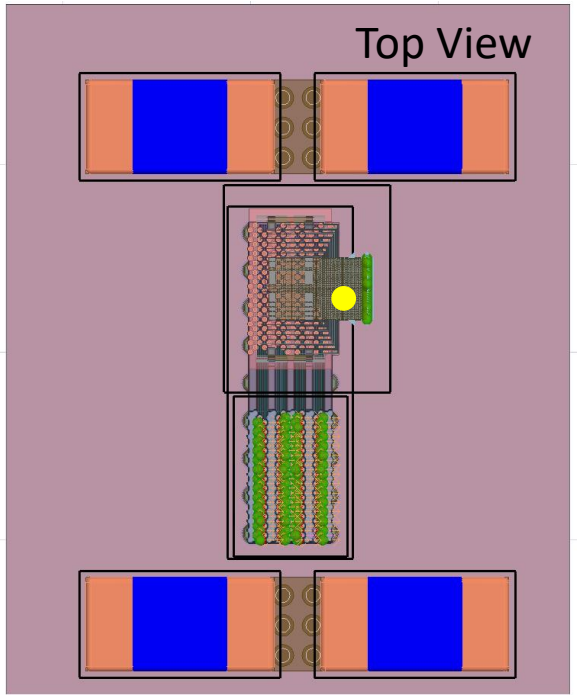


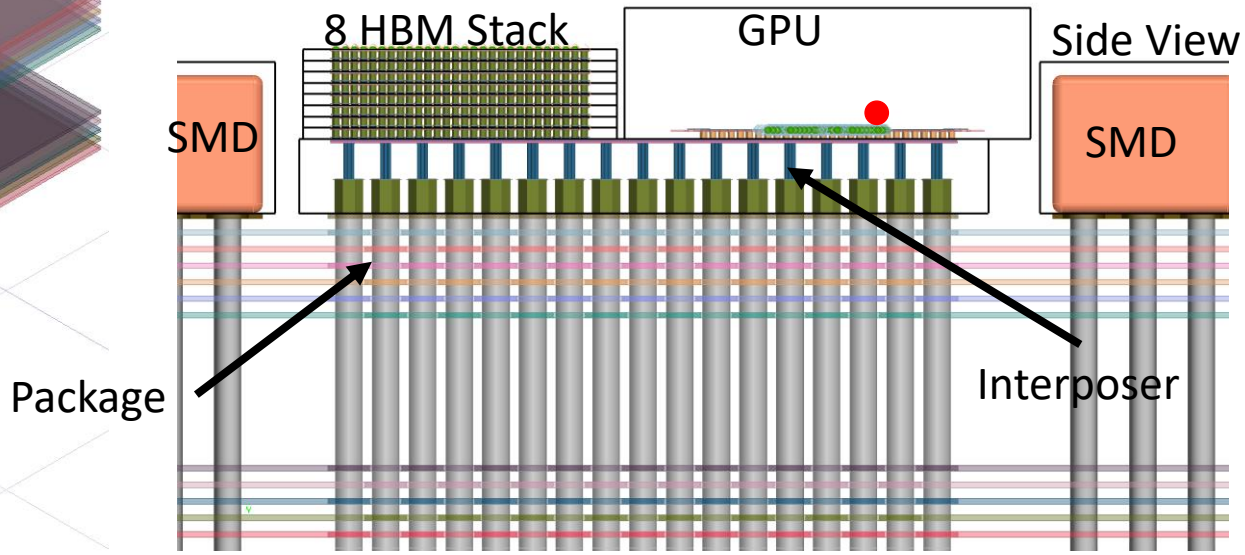
System Assembly 예제

System Assembly Example – 3DIC

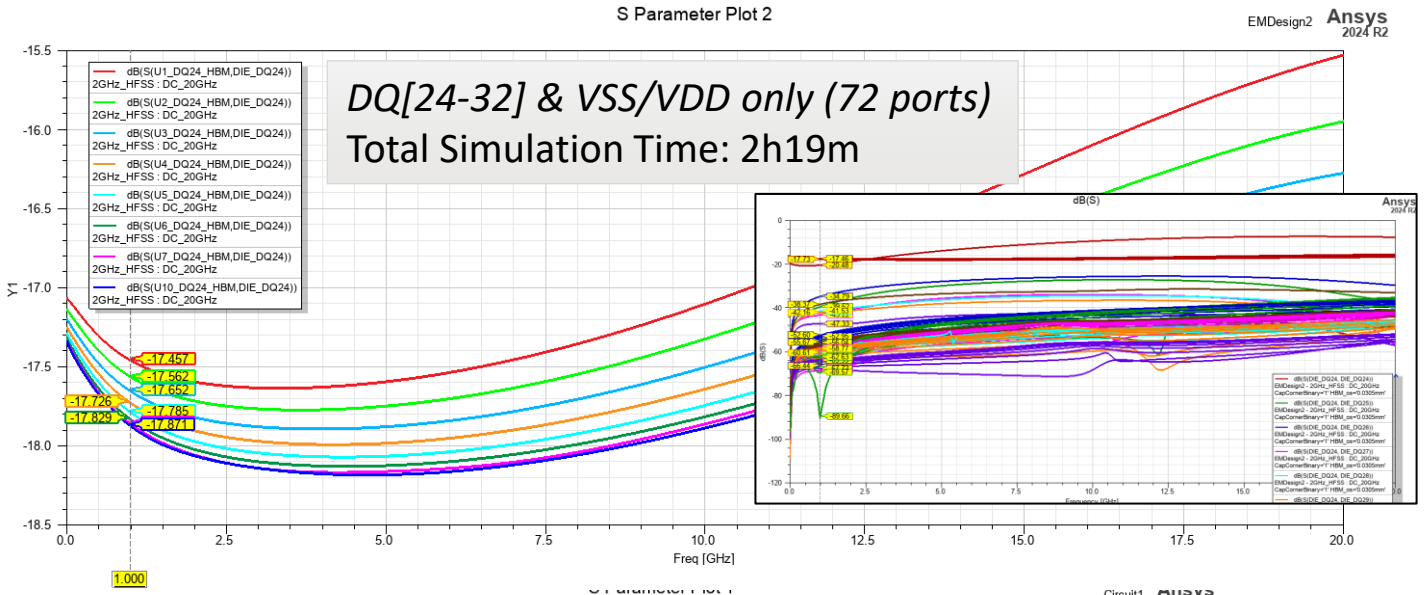
- GPU + HBM stack + SMD + Interposer



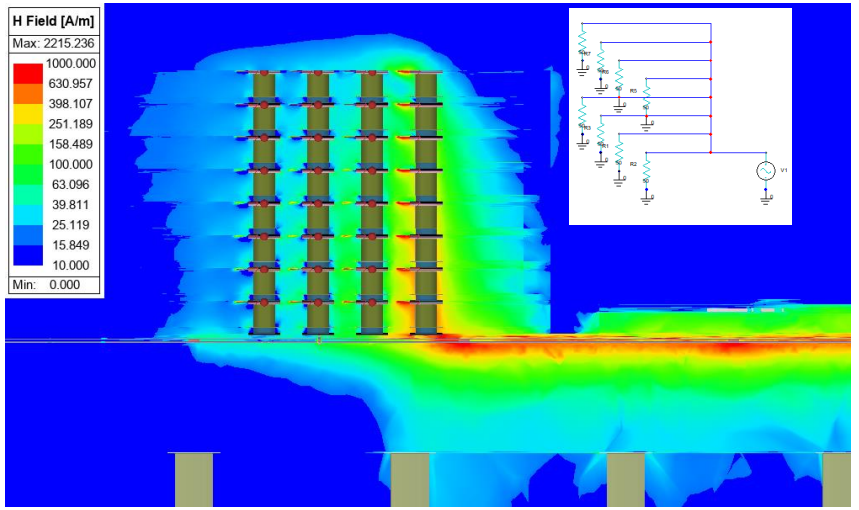


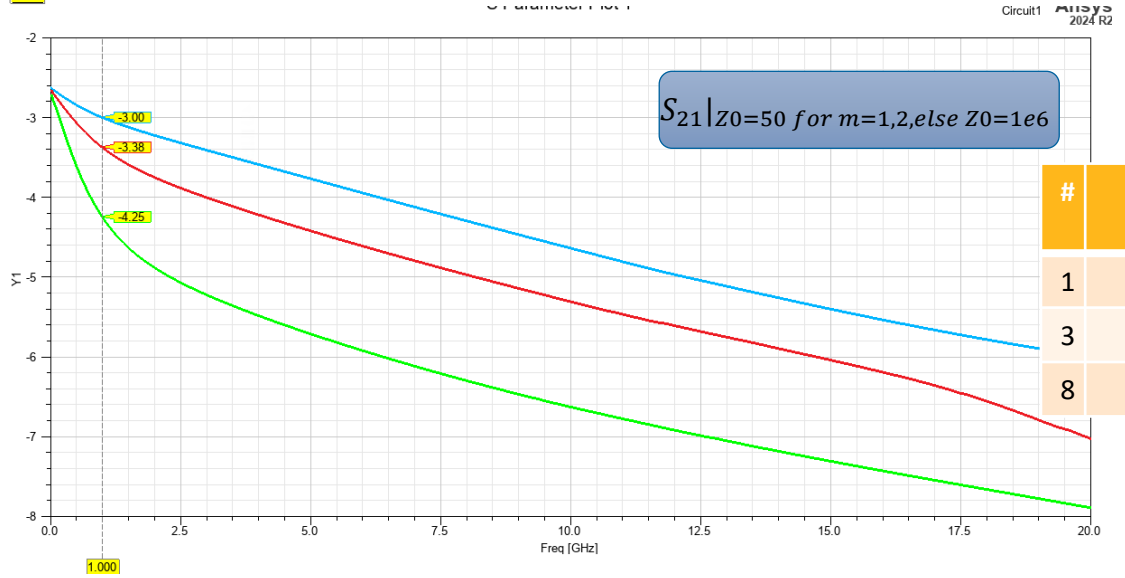


3DIC System: 8 stacked memory 해석 결과



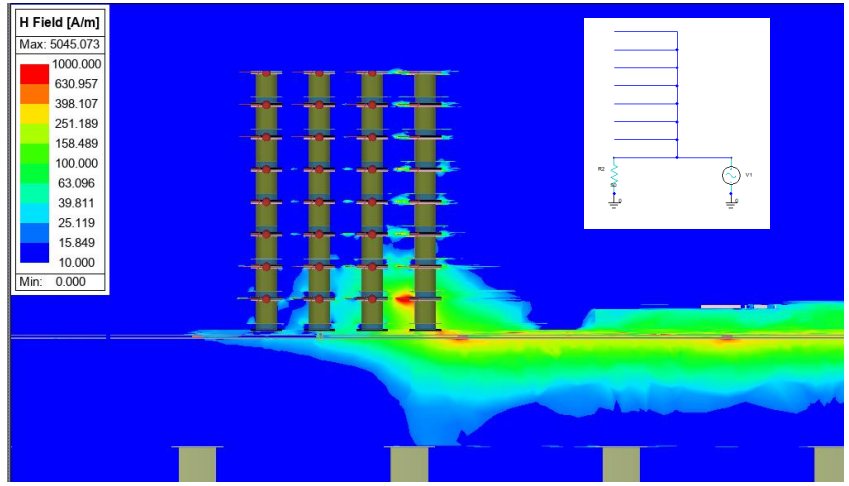
DQ[24-32] & VSS/VDD only (72 ports)
Total Simulation Time: 2h19m





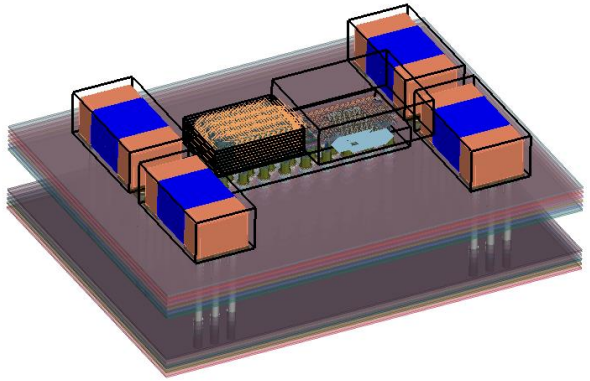
$S_{21}|_{Z0=50\ for\ m=1,2,else\ Z0=1e6}$

#	S_{21} DQ24(1GHz)
1	-3.00
3	-3.38
8	-4.25

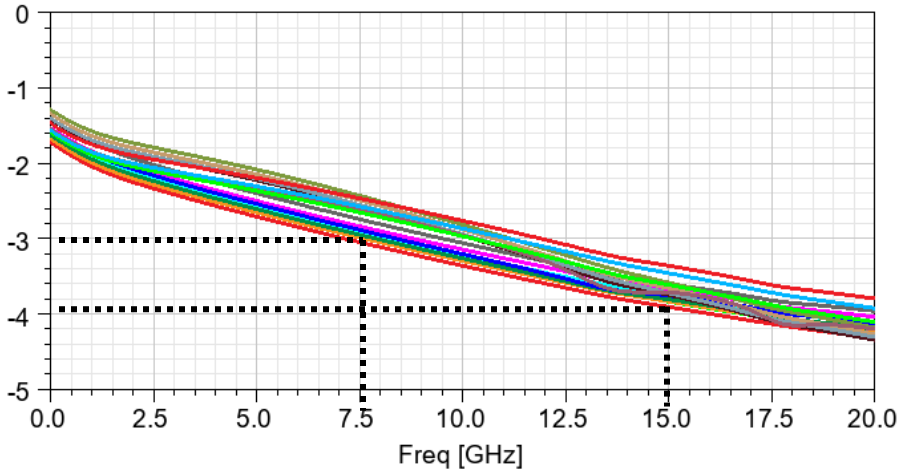


System Assembly vs. Cascaded S-parameter

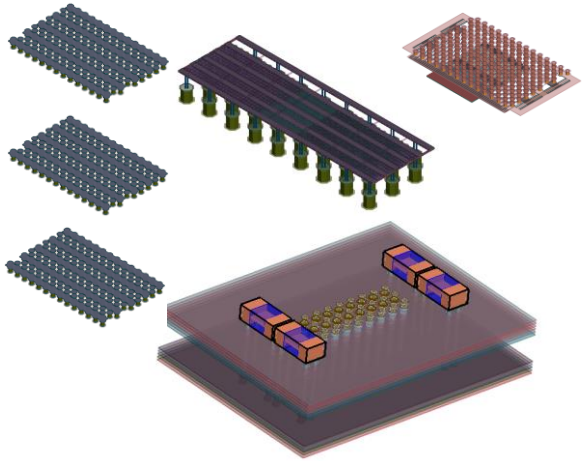
Fully EM coupled System Assembly

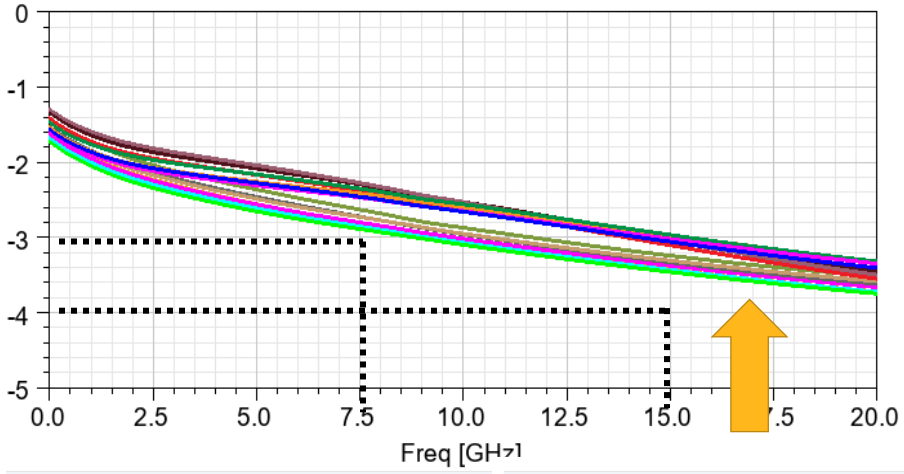


DQ Nets Insertion Loss (dB)



component별 S파라미터를 직렬로 연결한 경우

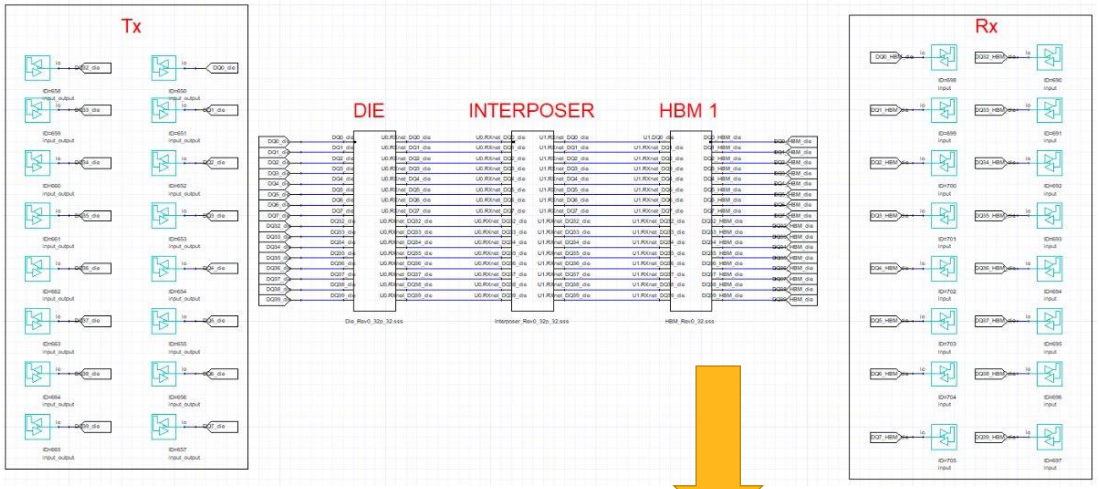


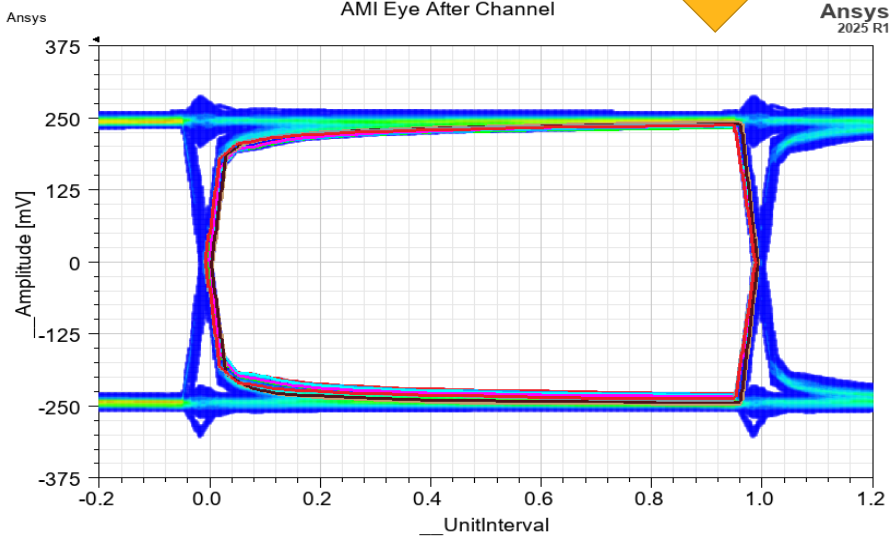


S21이 실제보다 좋게 나옴

System Assembly vs. Cascaded S-parameter (1xHBM)

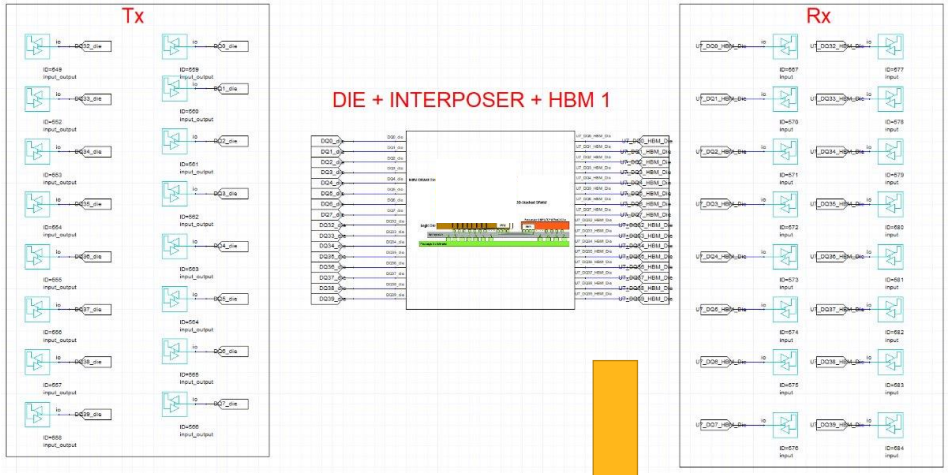
Cascade [S]

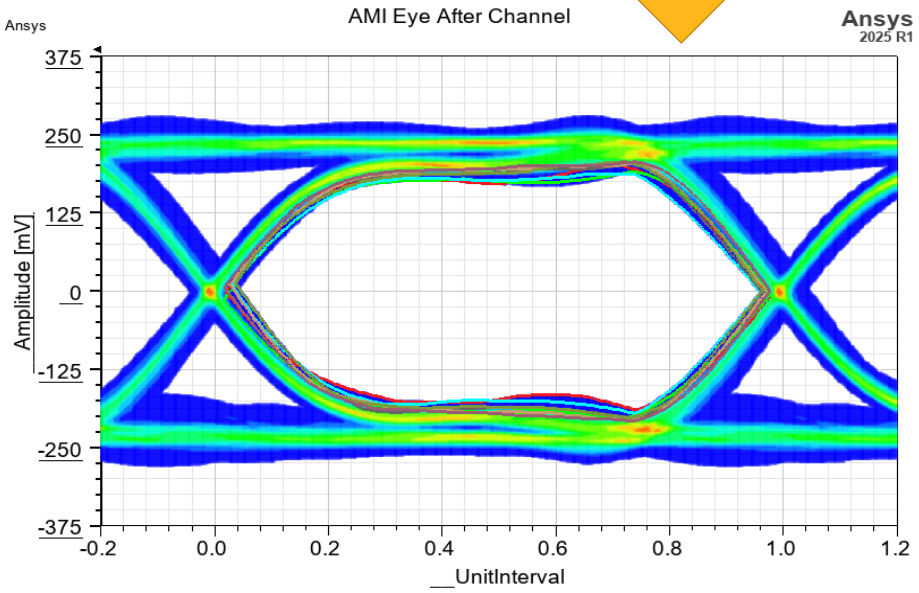




	MinEyeWidth	MinEyeHeight
DQ0 AMIAnalysis	0.9940	448.9427
DQ1 AMIAnalysis	0.9960	453.2961
DQ2 AMIAnalysis	0.9900	452.2366
DQ3 AMIAnalysis	0.9920	452.3769
DQ5 AMIAnalysis	0.9900	455.7284
DQ4 AMIAnalysis	0.9900	455.4319
DQ6 AMIAnalysis	0.9900	459.4760
DQ7 AMIAnalysis	0.9920	463.0726
DQ32 AMIAnalysis	0.9940	475.7104
DQ33 AMIAnalysis	0.9900	479.3295
DQ34 AMIAnalysis	0.9920	471.1189
DQ35 AMIAnalysis	0.9900	473.4059
DQ36 AMIAnalysis	0.9960	466.0771
DQ37 AMIAnalysis	0.9920	472.6779
DQ38 AMIAnalysis	0.9960	463.8334
DQ39 AMIAnalysis	0.9940	464.2655

Fully coupled EM solution

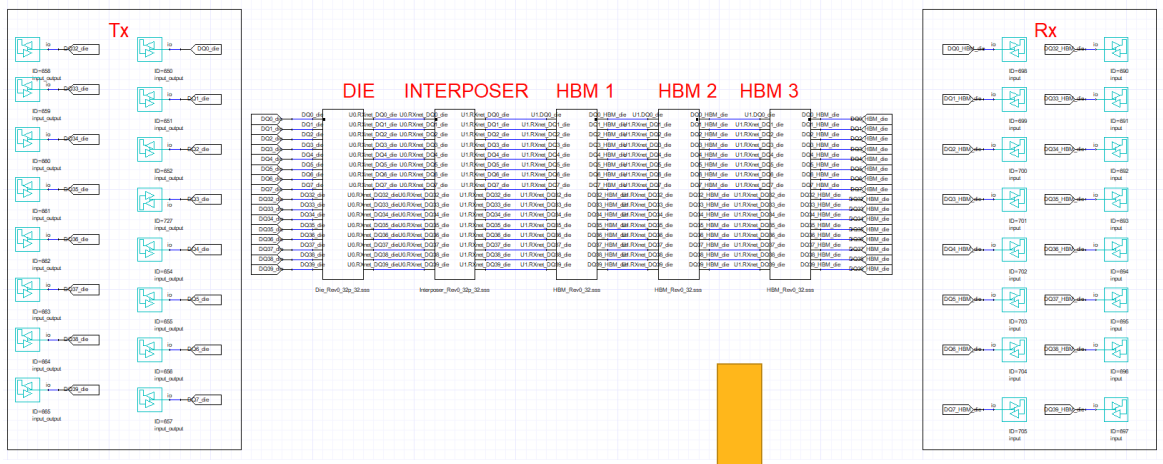


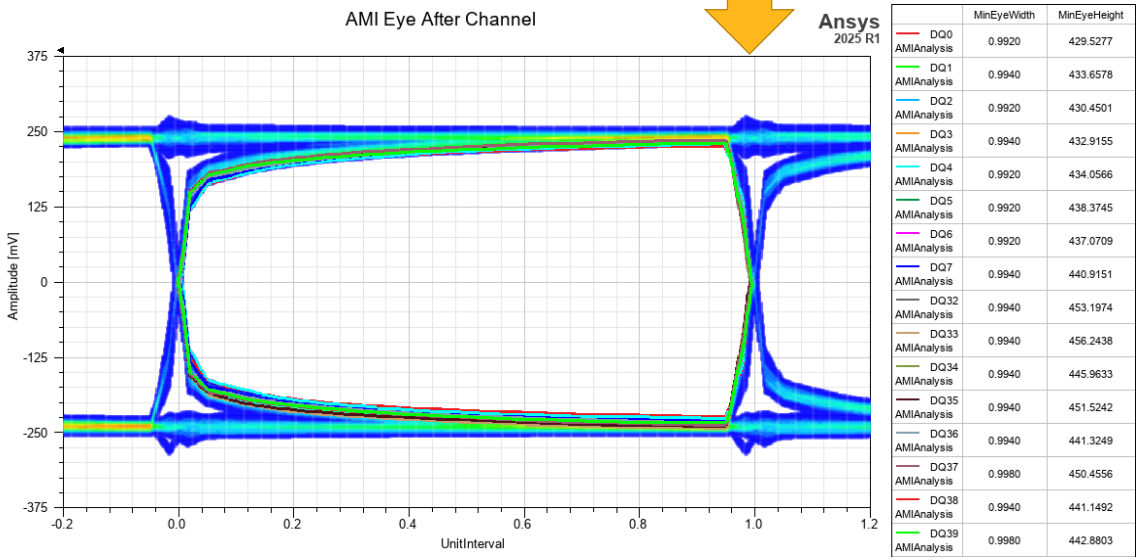


	MinEyeWidth	MinEyeHeight
DQ0 AMIAnalysis	0.9501	344.8091
DQ1 AMIAnalysis	0.9581	359.7657
DQ2 AMIAnalysis	0.9361	348.2145
DQ3 AMIAnalysis	0.9301	355.5077
DQ4 AMIAnalysis	0.9222	352.9096
DQ5 AMIAnalysis	0.9281	361.6559
DQ6 AMIAnalysis	0.9361	364.3663
DQ7 AMIAnalysis	0.9341	368.1840
DQ32 AMIAnalysis	0.9162	375.0724
DQ33 AMIAnalysis	0.9381	387.6840
DQ34 AMIAnalysis	0.9202	373.7398
DQ35 AMIAnalysis	0.9361	374.4995
DQ36 AMIAnalysis	0.9481	375.4959
DQ37 AMIAnalysis	0.9561	391.4764
DQ38 AMIAnalysis	0.9541	373.2327
DQ39 AMIAnalysis	0.9581	380.9468

System Assembly vs. Cascaded S-parameter (3xHBM)

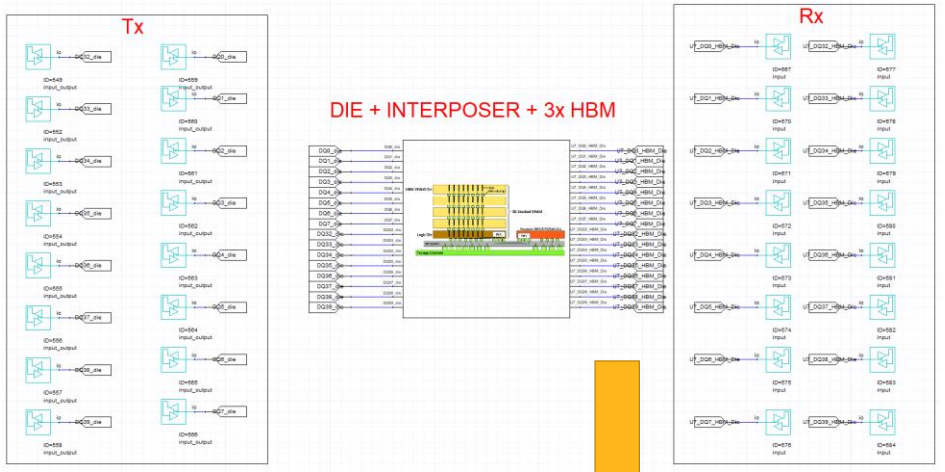
Cascade [S]

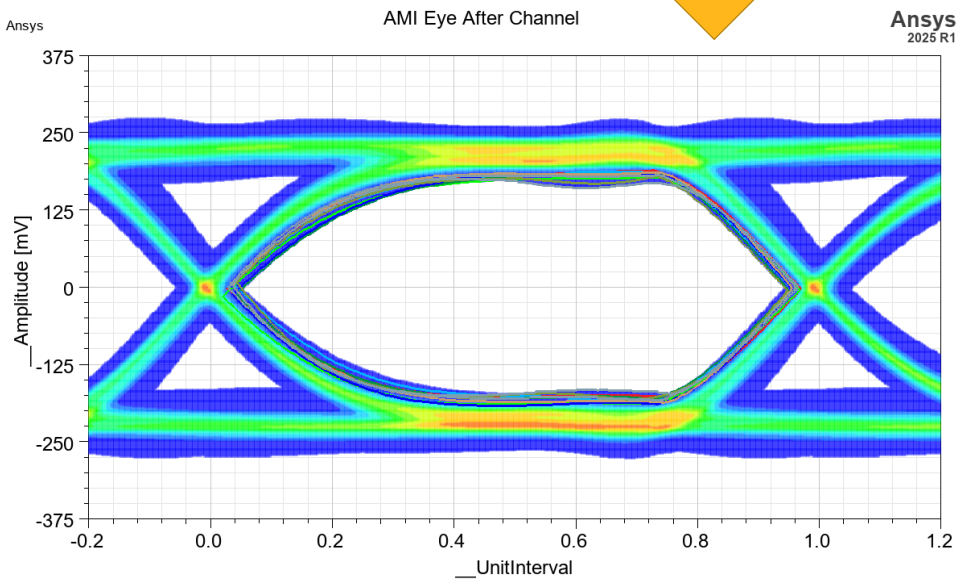




	MinEyeWidth	MinEyeHeight
DQ0 AMIAnalysis	0.9920	429.5277
DQ1 AMIAnalysis	0.9940	433.6578
DQ2 AMIAnalysis	0.9920	430.4501
DQ3 AMIAnalysis	0.9940	432.9155
DQ4 AMIAnalysis	0.9920	434.0566
DQ5 AMIAnalysis	0.9920	438.3745
DQ6 AMIAnalysis	0.9920	437.0709
DQ7 AMIAnalysis	0.9940	440.9151
DQ32 AMIAnalysis	0.9940	453.1974
DQ33 AMIAnalysis	0.9940	456.2438
DQ34 AMIAnalysis	0.9940	445.9633
DQ35 AMIAnalysis	0.9940	451.5242
DQ36 AMIAnalysis	0.9940	441.3249
DQ37 AMIAnalysis	0.9980	450.4556
DQ38 AMIAnalysis	0.9940	441.1492
DQ39 AMIAnalysis	0.9980	442.8803

Fully coupled EM solution





	MinEyeWidth	MinEyeHeight
DQ0 AMIAnalysis	0.9381	359.0356
DQ1 AMIAnalysis	0.9401	362.2057
DQ2 AMIAnalysis	0.9162	354.9886
DQ3 AMIAnalysis	0.9102	361.8827
DQ4 AMIAnalysis	0.9002	347.7672
DQ5 AMIAnalysis	0.8962	359.2896
DQ6 AMIAnalysis	0.9162	362.4101
DQ7 AMIAnalysis	0.9202	374.2859
DQ32 AMIAnalysis	0.8962	353.7648
DQ33 AMIAnalysis	0.9222	369.9815
DQ34 AMIAnalysis	0.8982	349.8421
DQ35 AMIAnalysis	0.9242	353.7712
DQ36 AMIAnalysis	0.9301	344.2358
DQ37 AMIAnalysis	0.9341	363.3830
DQ38 AMIAnalysis	0.9341	350.3478
DQ39 AMIAnalysis	0.9341	358.3465

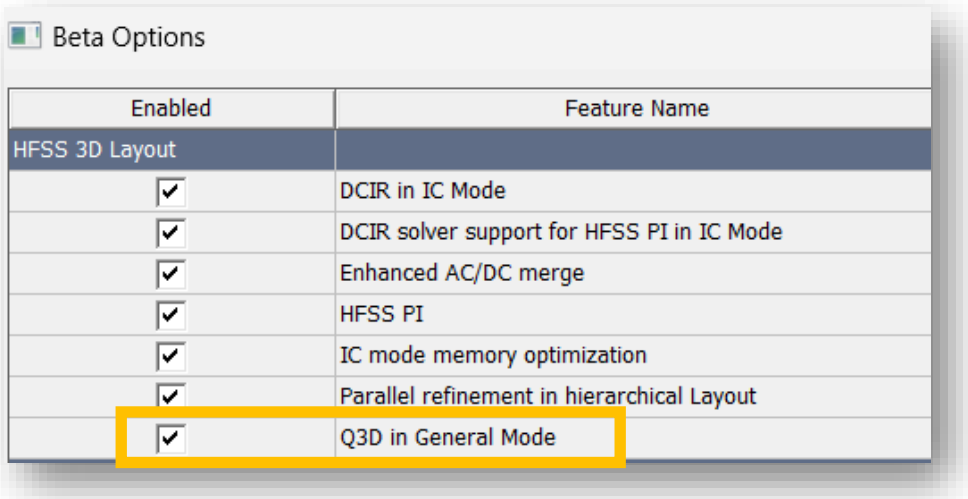
결론 : System Assembly vs. Cascaded S-parameter

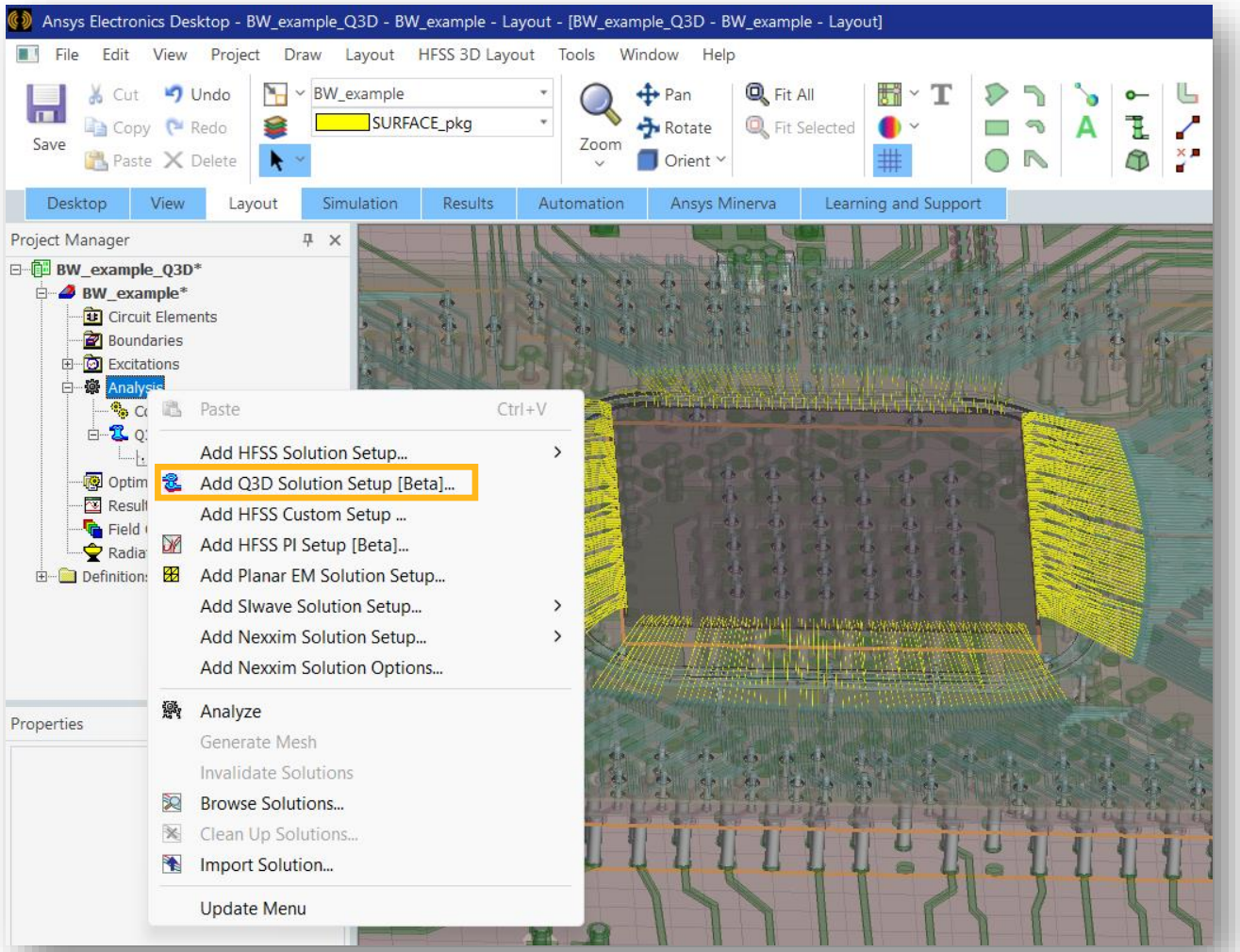
- 해석의 편의를 위해, 신호 채널의 여러 component 를 따로 해석한 S 파라미터를 연결하여 전체 시스템 성능을 예측하기도 함.
- 이러한 Cascaded S-parameter 방법은 빠르고 효율적이긴 하지만, 각 component 간의 EM coupling으로 인한 손실이 고려되지 않음
- HFSS 또는 HFSS 3D Layout 기반의 System Assembly 해석은 component 배치에 따른 EM coupling을 보다 정확히 계산할 수 있음

2. Q3D in HFSS 3D Layout

Q3D in HFSS 3D Layout general mode (Beta)

- Package 및 PCB 추출을 위한 IC 모드와 일반 모드 모두에서 사용 가능한 Q3D 솔버
- System assembly 구조도 해석 가능
- **RLC component 지원**

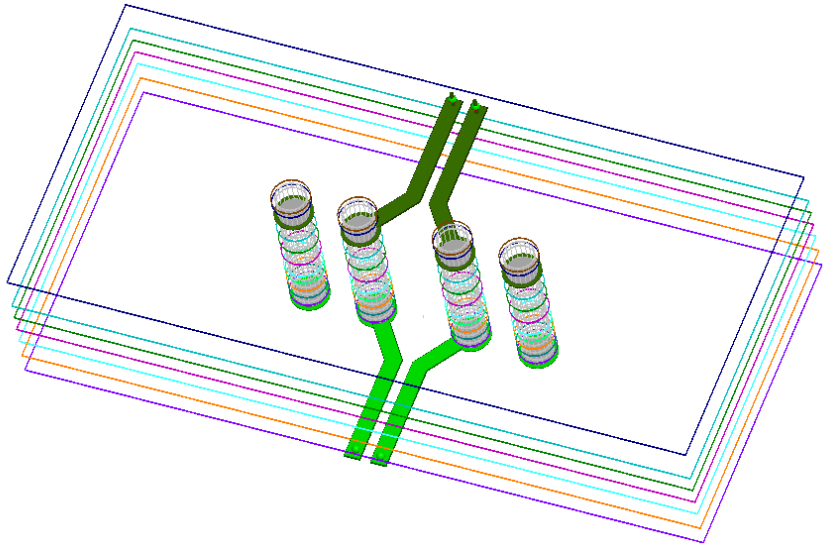


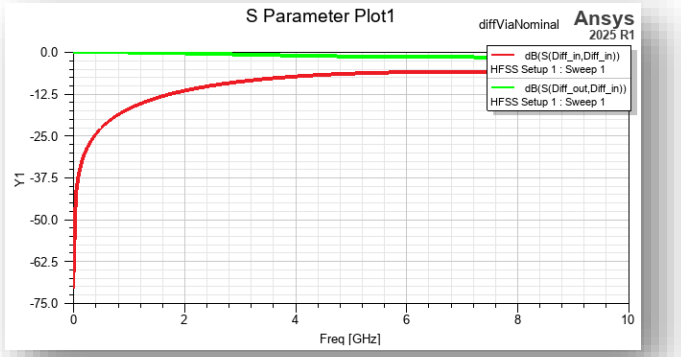


HFSS 3D Layout - Q3D in General Mode의 장점

1. HFSS 를 위한 모델과 셋업 그대로 즉시 Q3D solver를 돌릴 수 있음
   - 별도로 Q3D export 및 source/sink modeling이 필요 없음
2. HFSS 3D Layout 특유의 빠르고 가벼운 GUI를 활용할 수 있음
   - 복잡하고 큰 모델의 경우, 기존의 Q3D보다 작업효율이 크게 개선됨
3. RLC component를 포함한 RLGC 추출이 가능 (기존 Q3D는 불가능)

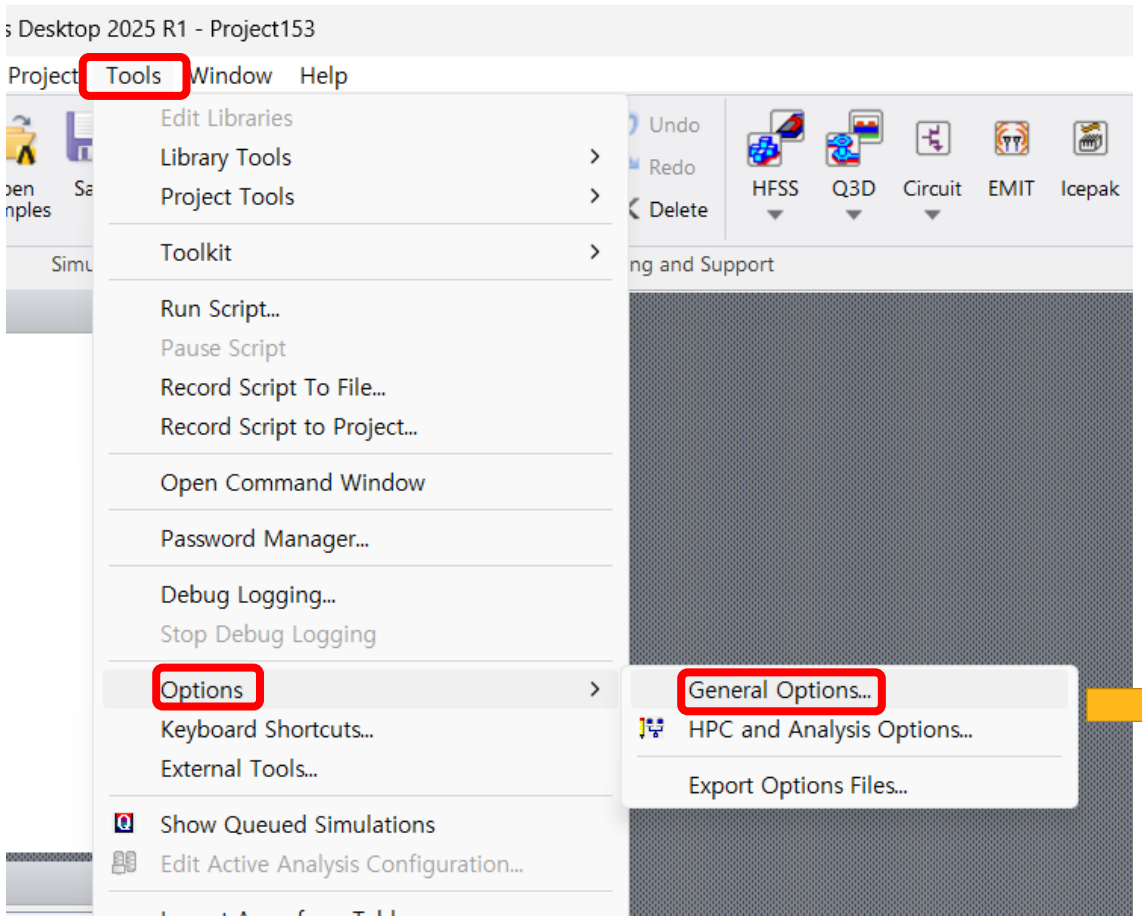
→ 동일한 모델로 한꺼번에 HFSS S parameter와 Q3D RLGC 데이터를 추출 가능

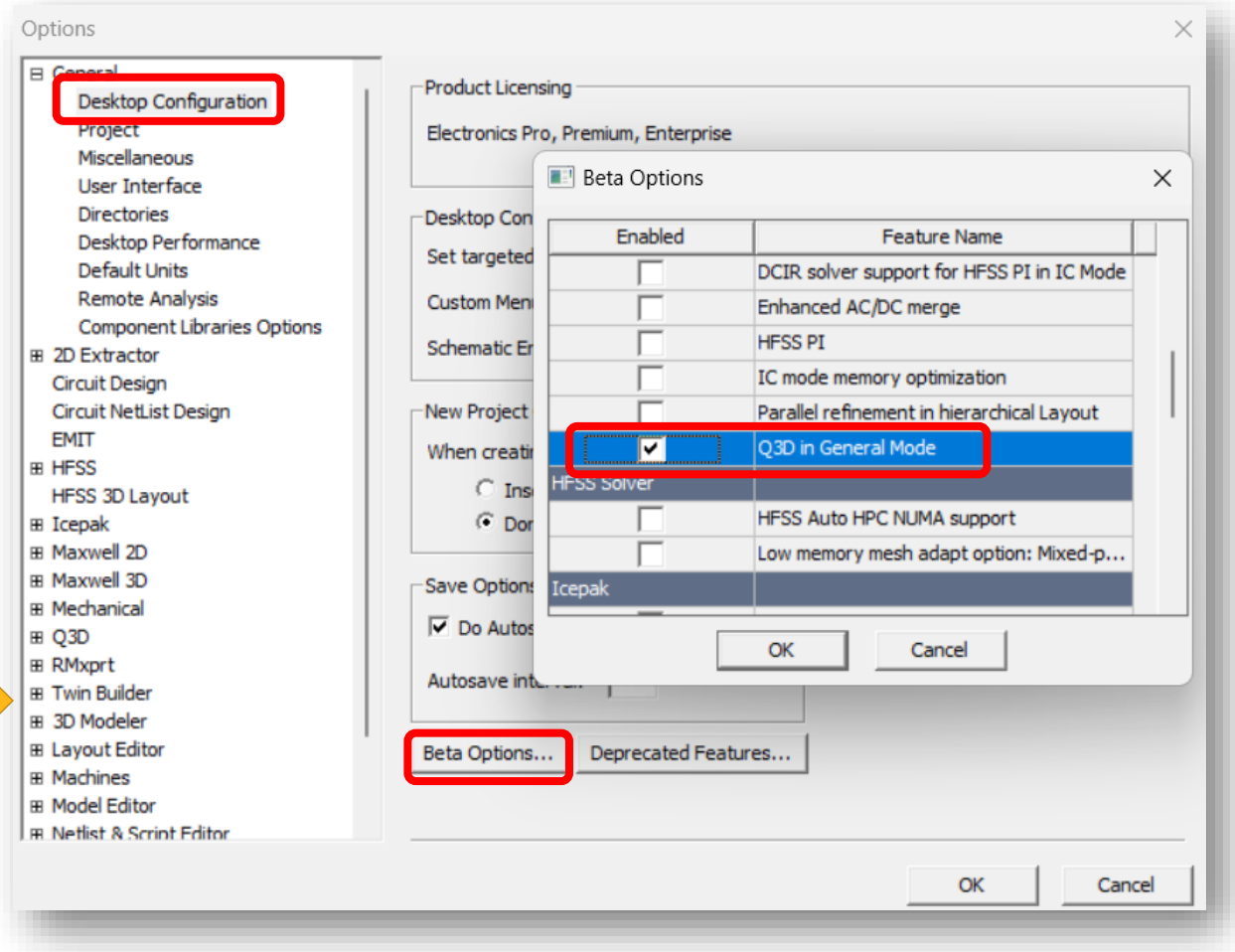




	Diff_N_in->Diff_N_out	Diff_N_out#ref->Diff_N_in#ref	Diff_P_in->Diff_P_out	Diff_P_in#ref->Diff_N_in#ref	Diff_P_out#ref->Diff_N_in#ref
Freq: 10MHz					
Diff_N_in->Diff_N_out	0.01984, 4.08272	0.00000, -1.38150	0.00000, 1.73896	0.00000, -0.02384	0.00000, -1.36027
Diff_N_out#ref->Diff_N_in#ref	0.00000, -1.38150	0.00285, 1.87285	0.00000, -1.33774	0.00062, 0.03928	0.00231, 1.82890
Diff_P_in->Diff_P_out	0.00000, 1.73896	0.00000, -1.33774	0.01973, 4.07026	0.00000, 0.03364	0.00000, -1.34764
Diff_P_in#ref->Diff_N_in#ref	0.00000, -0.02384	0.00062, 0.03928	0.00000, 0.03364	0.00124, 0.09188	0.00062, 0.04322
Diff_P_out#ref->Diff_N_in#ref	0.00000, -1.36027	0.00231, 1.82890	0.00000, -1.34764	0.00062, 0.04322	0.00279, 1.86597

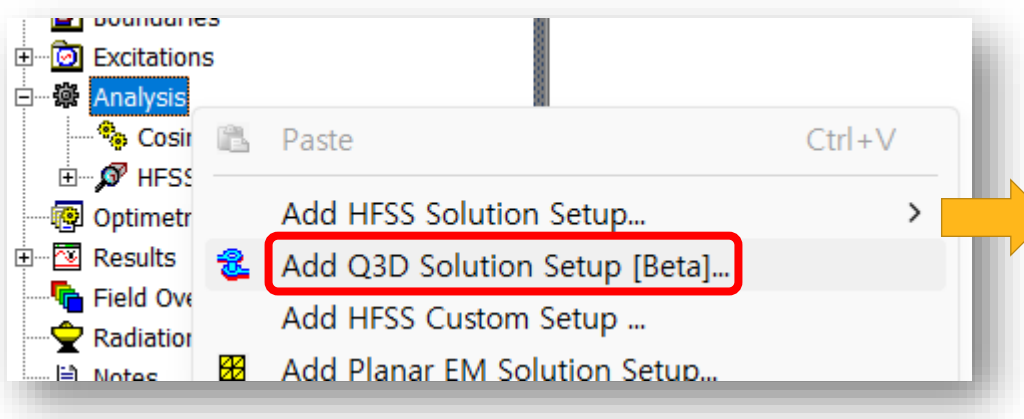
Q3D in General Mode Setup (Beta)



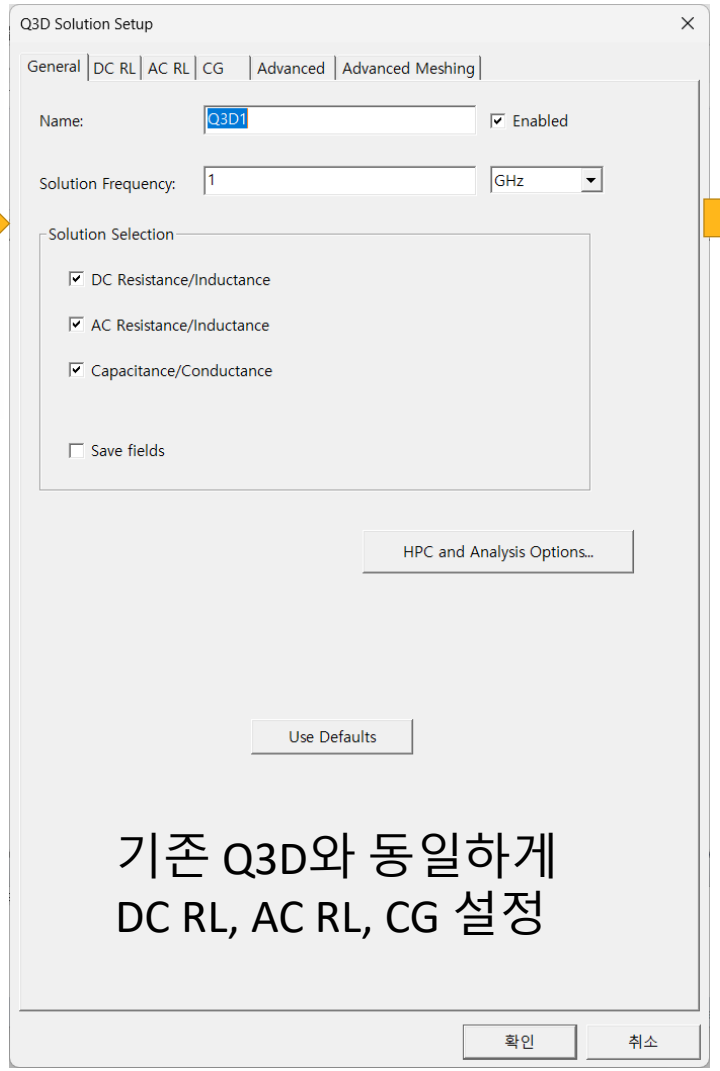


- Beta 기능으로서, Option에서 활성화시켜야 사용 가능

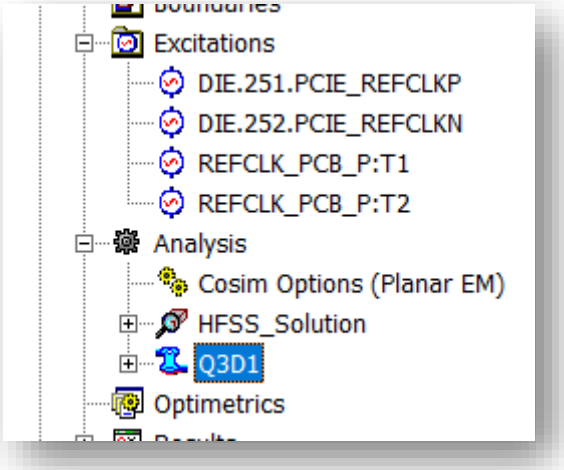
HFSS 3D Layout - Q3D Setup 추가



Analysis에서 Q3D setup 추가

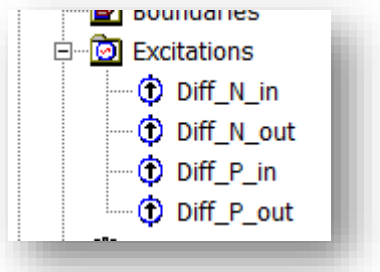


기존 Q3D와 동일하게
DC RL, AC RL, CG 설정

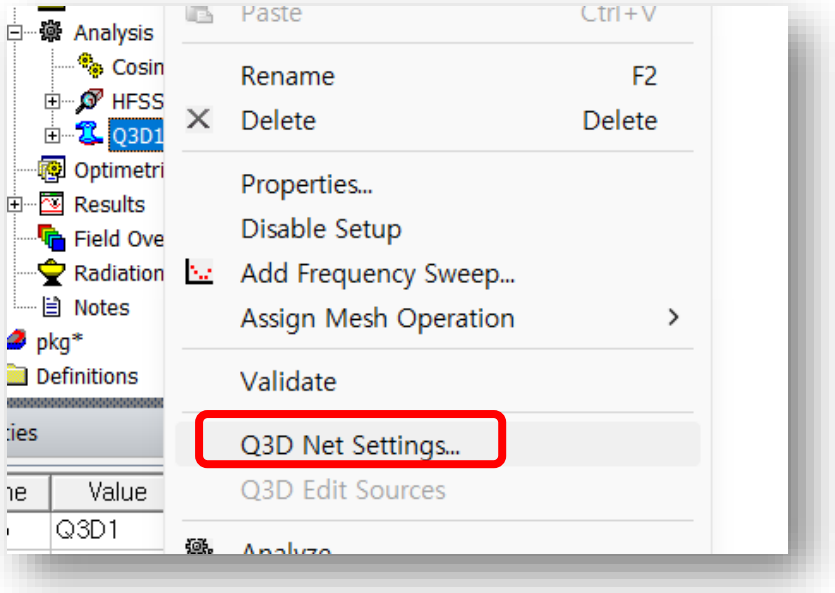


Q3D setup 생성

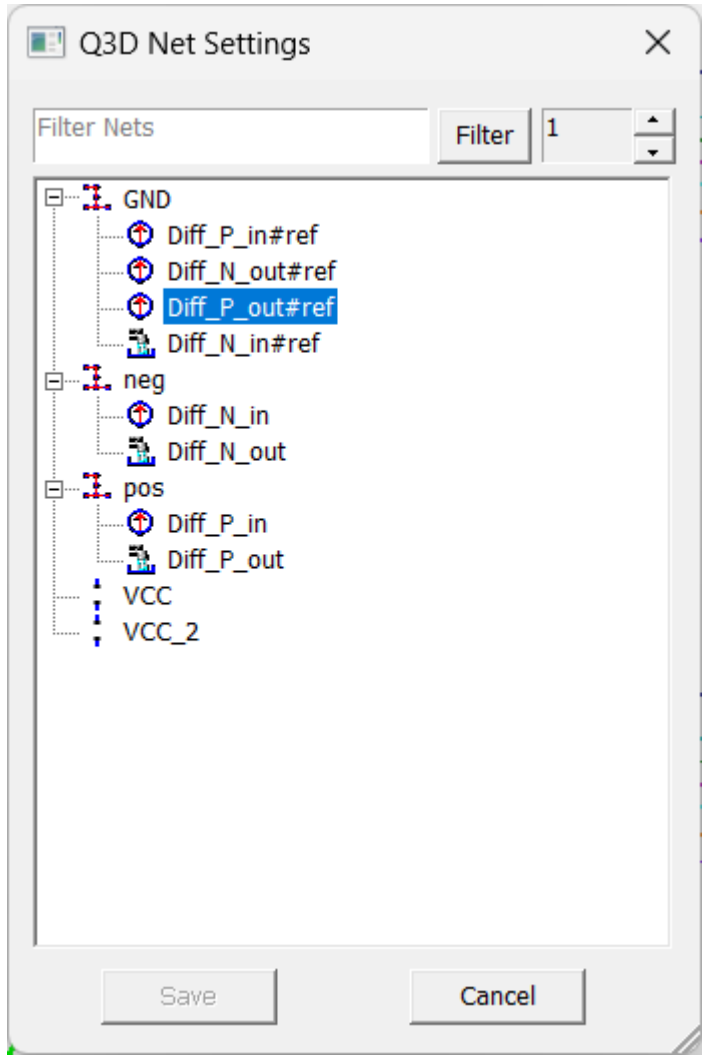
Q3D net (source/sink) 설정 in HFSS 3D Layout



HFSS Port 설정

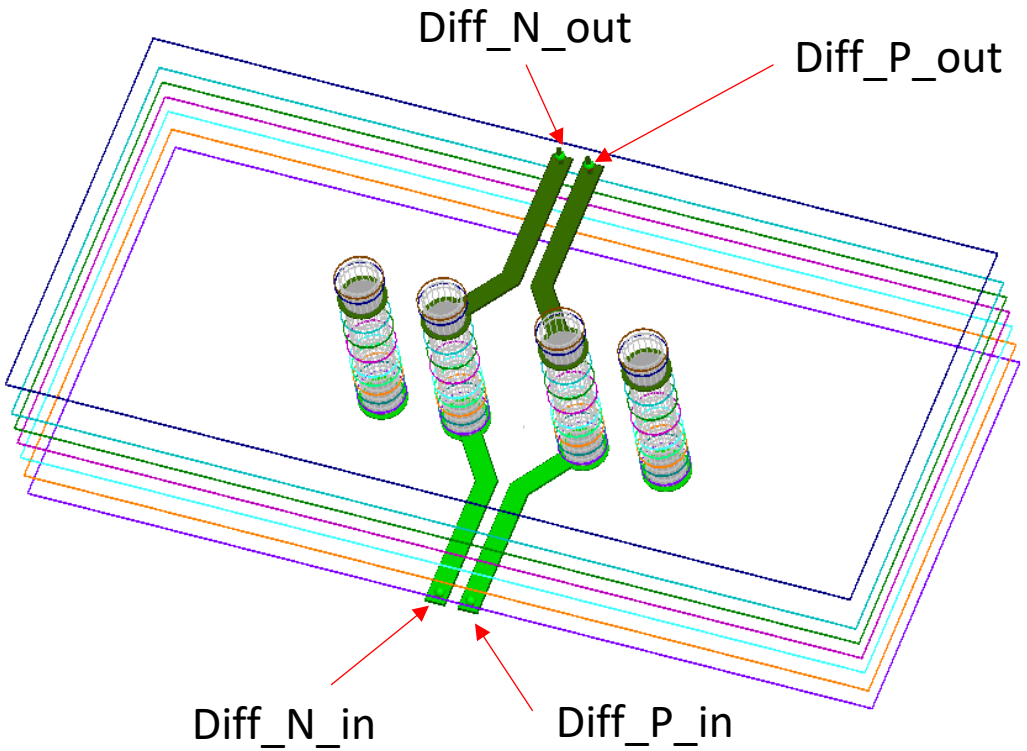


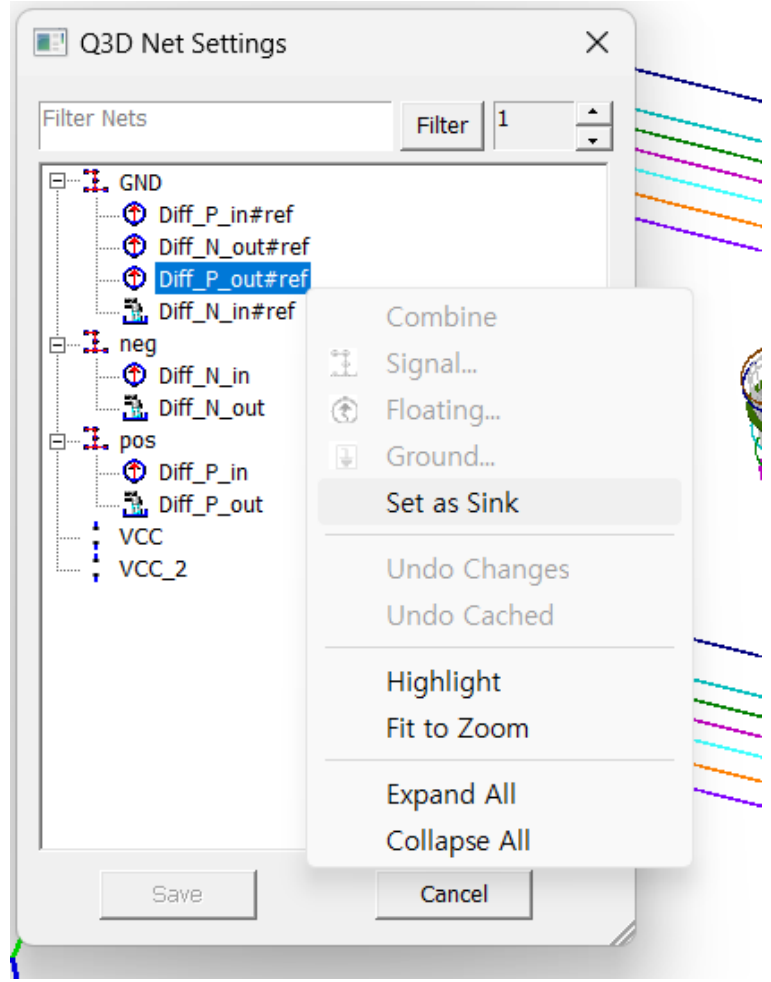
Q3D setup에서 우클릭
→ Q3D Net Seetings



HFSS Port를 기준으로 자동으로
source/sink와 net이 설정됨

Automatic Source/Sink Assign & modifying



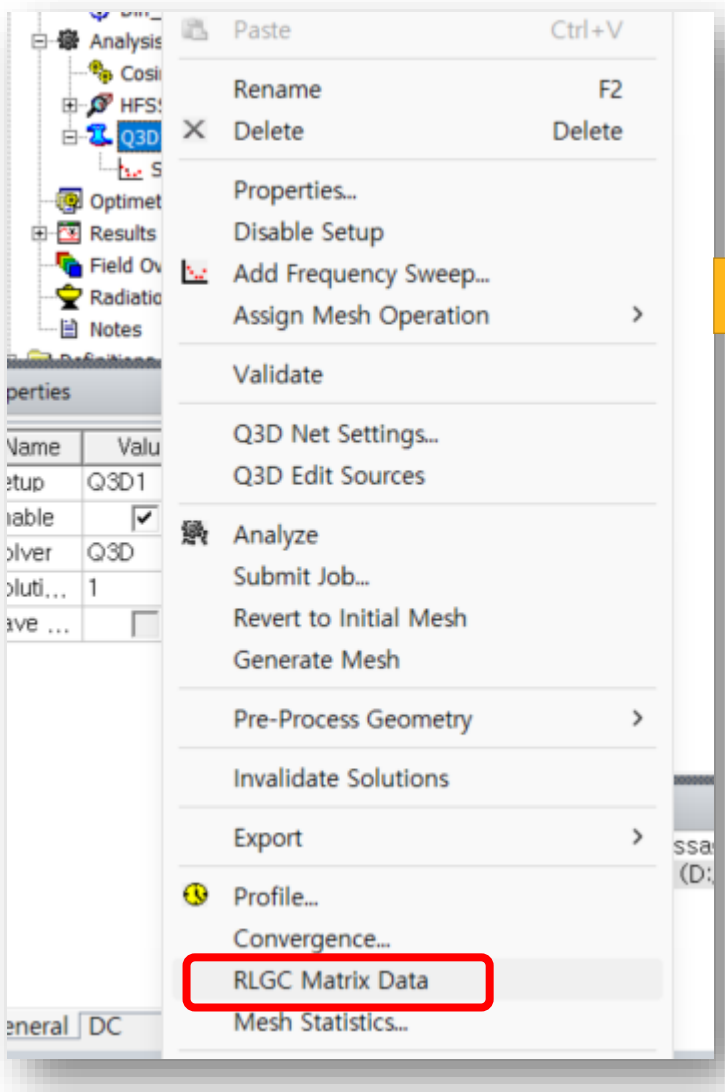


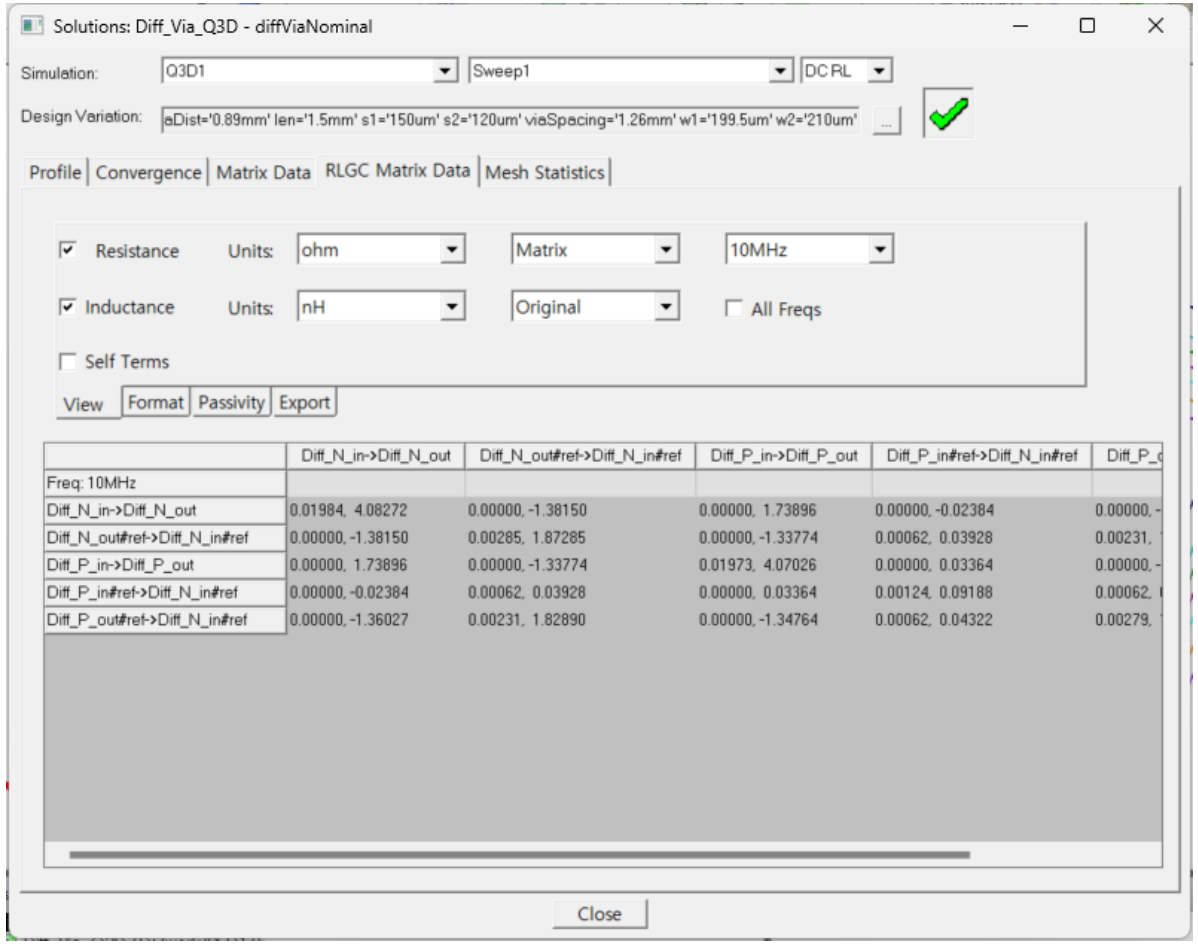
미리 생성된 HFSS Port 를 기준으로
→ Signal net 시작과 끝에 자동으로 source/sink 배치
→ 공통 Reference net에 1개의 sink와 나머지는 source 자동 배정

- Wave port는 불가능, terminal과 lumped port에만 가능

User가 source/sink를 변경 가능

RLGC Result



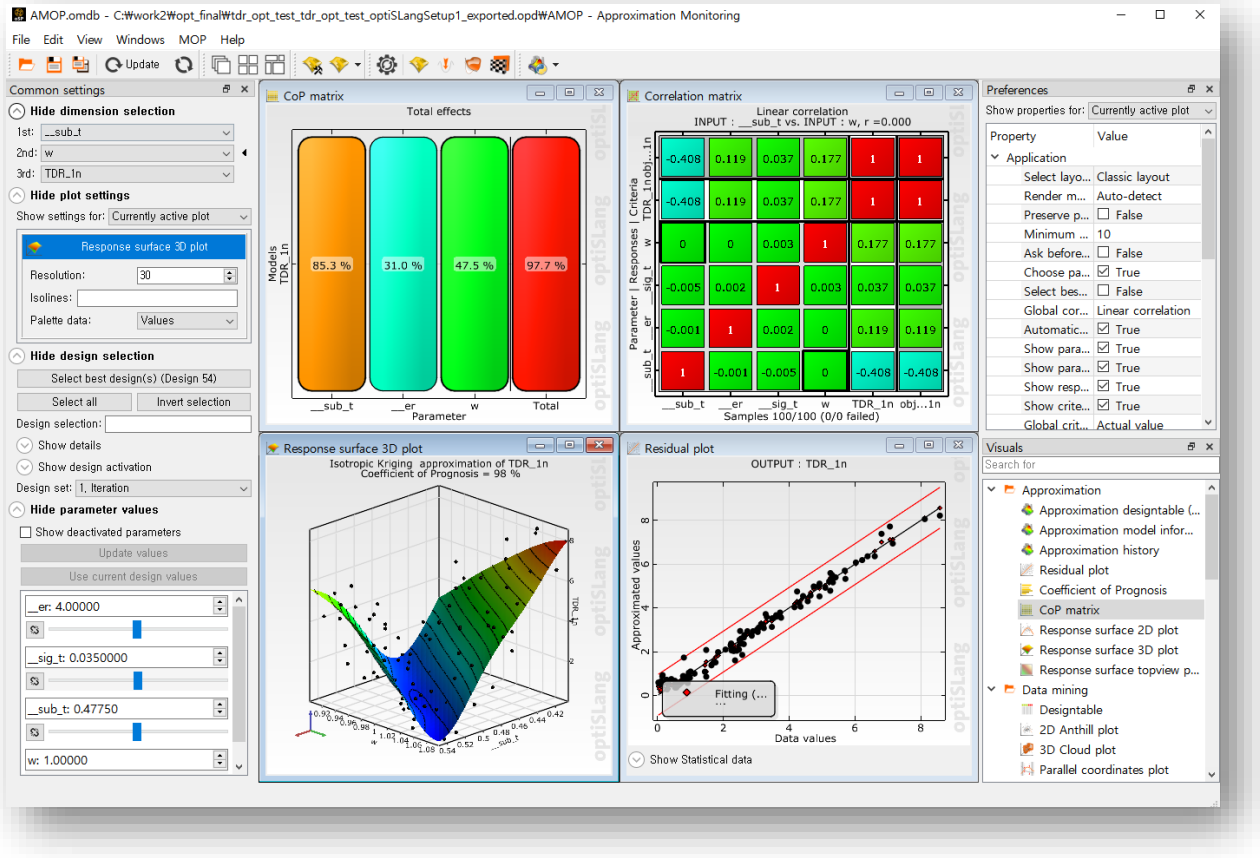


	Diff_N_in->Diff_N_out	Diff_N_out#ref->Diff_N_in#ref	Diff_P_in->Diff_P_out	Diff_P_in#ref->Diff_N_in#ref
Freq: 10MHz				
Diff_N_in->Diff_N_out	0.01984, 4.08272	0.00000, -1.38150	0.00000, 1.73896	0.00000, -0.02384
Diff_N_out#ref->Diff_N_in#ref	0.00000, -1.38150	0.00285, 1.87285	0.00000, -1.33774	0.00062, 0.03928
Diff_P_in->Diff_P_out	0.00000, 1.73896	0.00000, -1.33774	0.01973, 4.07026	0.00000, 0.03364
Diff_P_in#ref->Diff_N_in#ref	0.00000, -0.02384	0.00062, 0.03928	0.00000, 0.03364	0.00124, 0.09188
Diff_P_out#ref->Diff_N_in#ref	0.00000, -1.36027	0.00231, 1.82890	0.00000, -1.34764	0.00062, 0.04322

- RLGC 결과를 확인하는 방법은 기존 Q3D와 동일

3. Optislang – Metamodel Automation

Ansys optiSLang for Metamodel

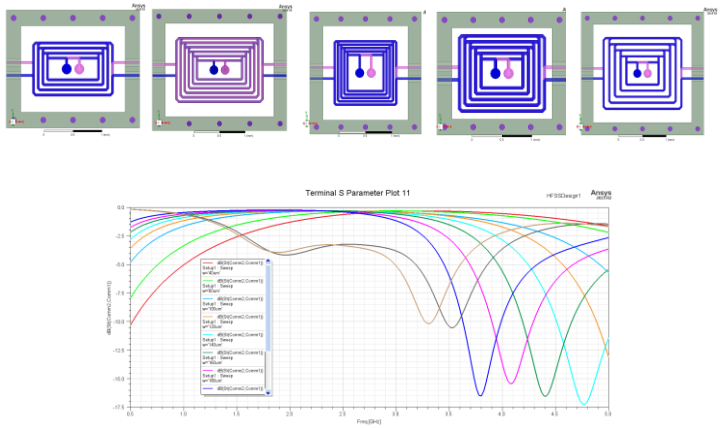


- Ansys의 고성능 최적화 툴 (별도 판매)
- Deep learning 을 포함한 다양한 알고리즘 탑재
- HFSS의 설계 변수를 직접 조절하면서 대량의 시뮬레이션 데이터를 자동으로 생성
- 해석된 HFSS 데이터를 기반으로 Metamodel 생성

→ Metamodel을 이용하면 목표성능 구현에 필요한 설계변수를 수 초 내에 즉시 계산해줌.
(사실상 설계용 GPT 같은 역할이라고 이해하면 됨)

Key point of optiSLang Metamodel

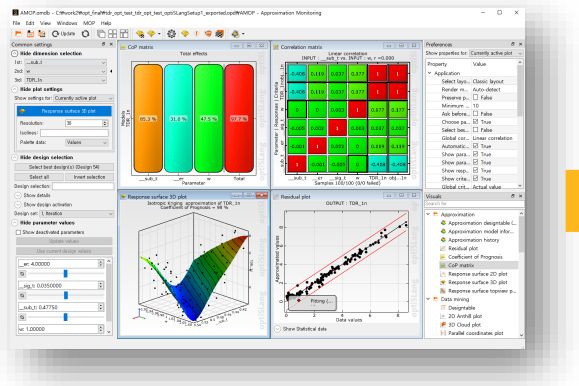
해석 데이터



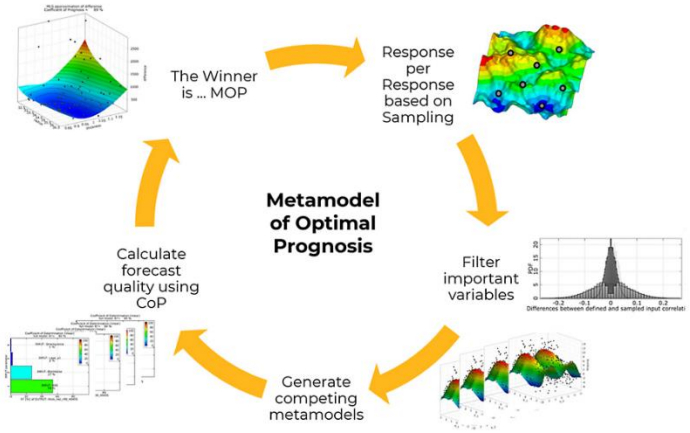
설계 변수 튜닝을 통한 대량의 시뮬레이션 데이터

훈련

optiSLang



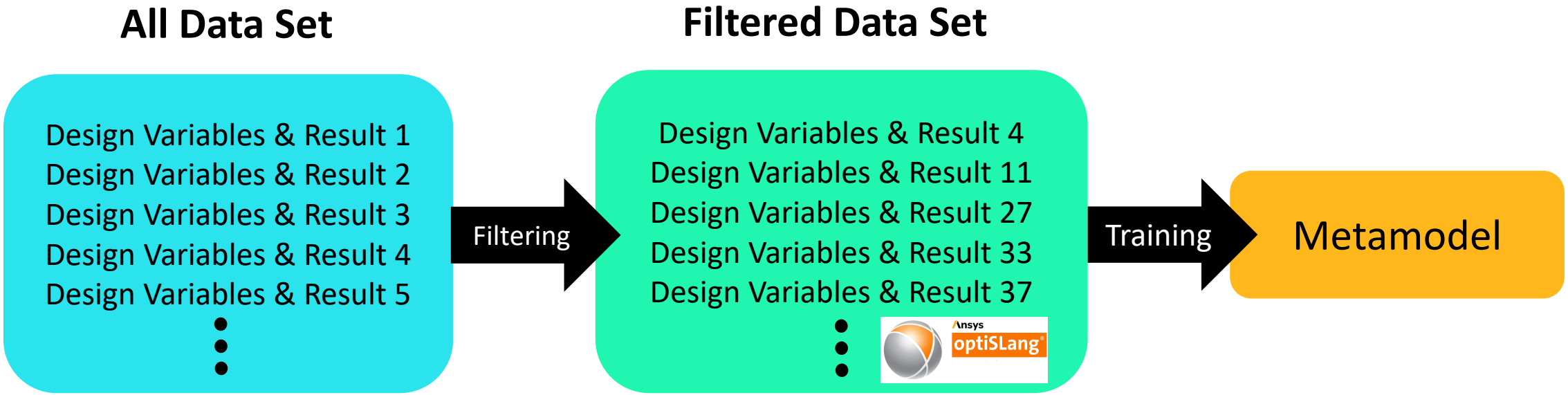
메타 모델



성능목표를 만족하는 최적 설계값을 제시

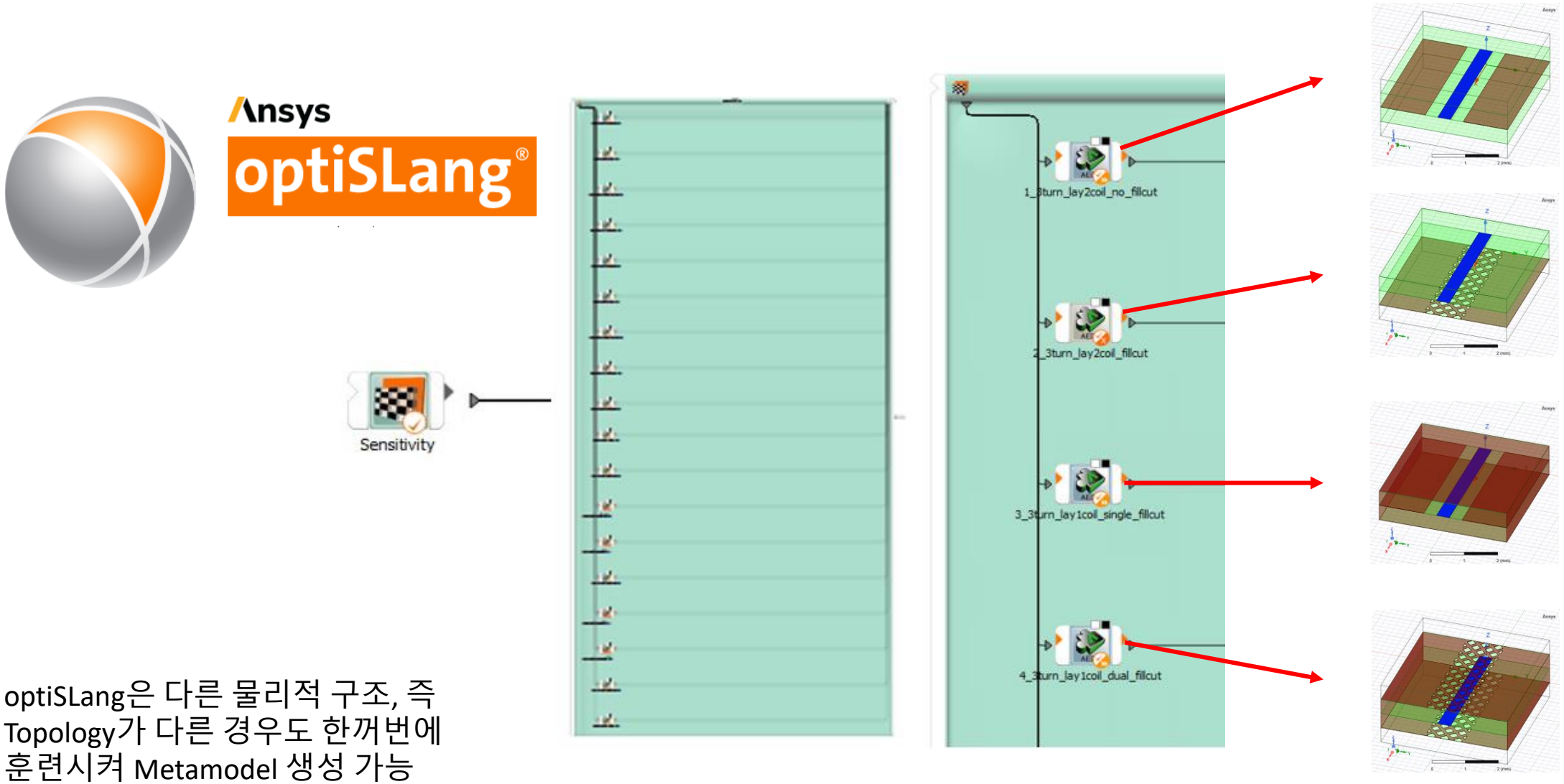
- Metamodel 기반 최적화는 Big data에 기반한 Machine Learning으로 AI를 구현하는 과정과 동일
- 대량의 설계 변수와 해석결과 set는 잘 정제된 big data로서 동작
- optiSLang의 알고리즘이 설계 변수(input)와 해석결과(output)와의 연관관계를 지능적으로 파악
- 이러한 훈련을 기반으로 목표 결과를 제시하면 즉시 설계변수를 추론해내는 Metamodel 생성

optiSLang : Filtered Data Set for Metamodel



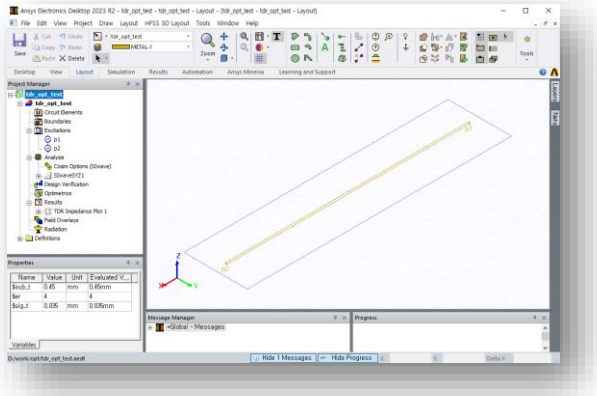
- optiSLang 은 목표 성능을 구현하는 입력 변수의 경향성을 빠르게 뽑아내고, 의미 있는 결과를 내는 변수 조합의 방향성을 찾아냄으로서 Data Filtering 효과가 탁월함.
- 10000개의 입력변수 조합 중 100개만 추려내어 metamodel을 만들어도, 10000개를 모두 사용한 경우와 비슷
- 불필요한 iteration 을 줄여서 데이터를 뽑아내는 해석시간을 크게 줄일 수 있음

optiSLang : **Multi-topology** Metamodel

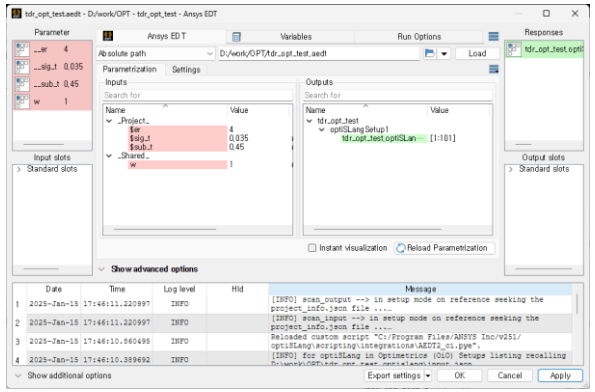


- optiSLang은 다른 물리적 구조, 즉 Topology가 다른 경우도 한꺼번에 훈련시켜 Metamodel 생성 가능

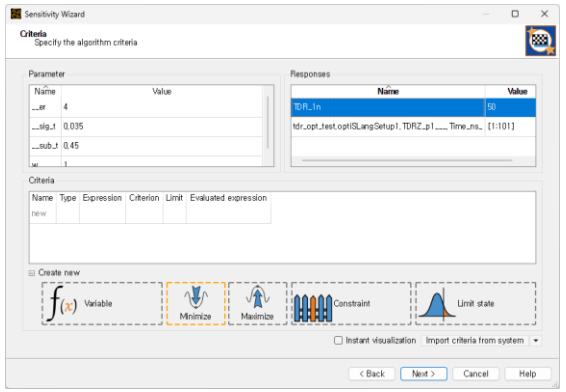
optiSLang Manual Process



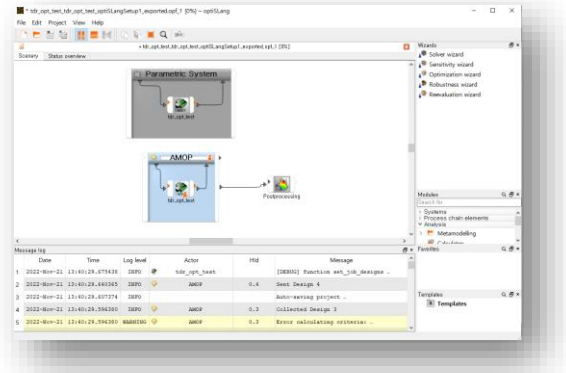
1 AEDT에서 설계 변수 설정 및 1차 해석결과 도출



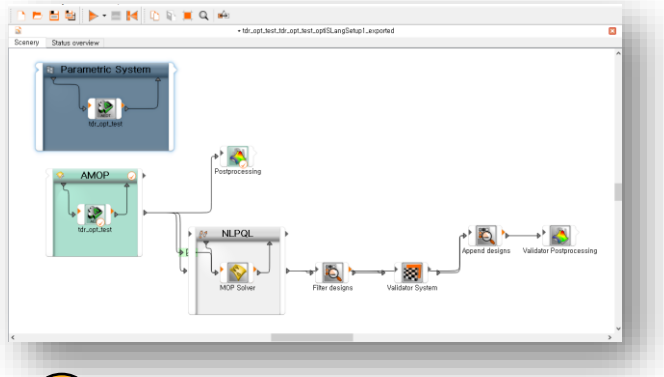
2 optiSLang에서 설계변수 및 해석결과 import



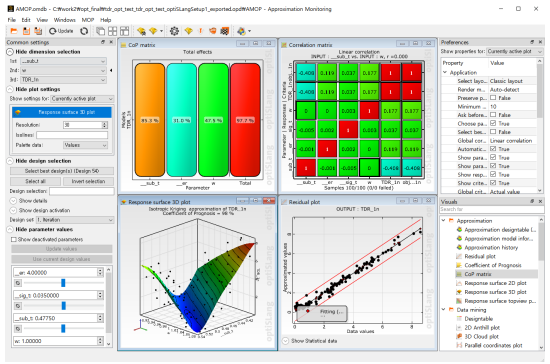
3 목표 성능 및 알고리즘 선택



4 Sensitivity analysis 수행으로 Metamodel 생성



5 Metamodel 기반 최적화 수행



6 결과 분석

optiSLang 의 어려움..

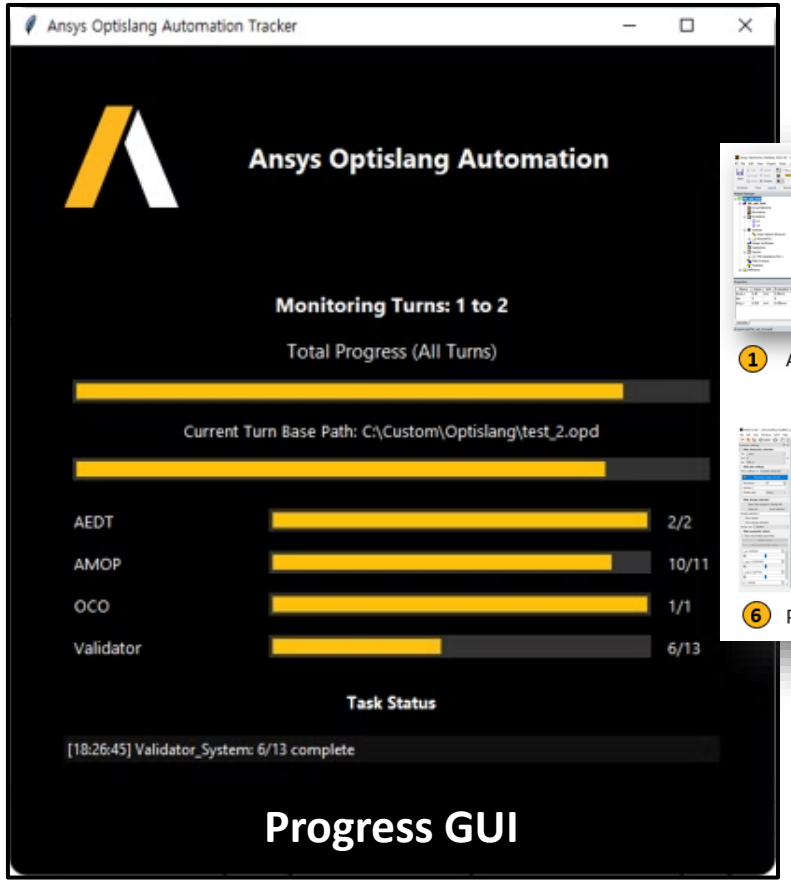
1. 전자 엔지니어에게 어렵게 느껴짐

- 최적화 엔지니어링이 중요한 기계과에서 유래된 툴
- 어색한 GUI , 생소한 진행과정과 어려운 최적화 전문용어들
- 작은 사용자 실수에도 에러 발생 (정교하지만 예민한 알고리즘)

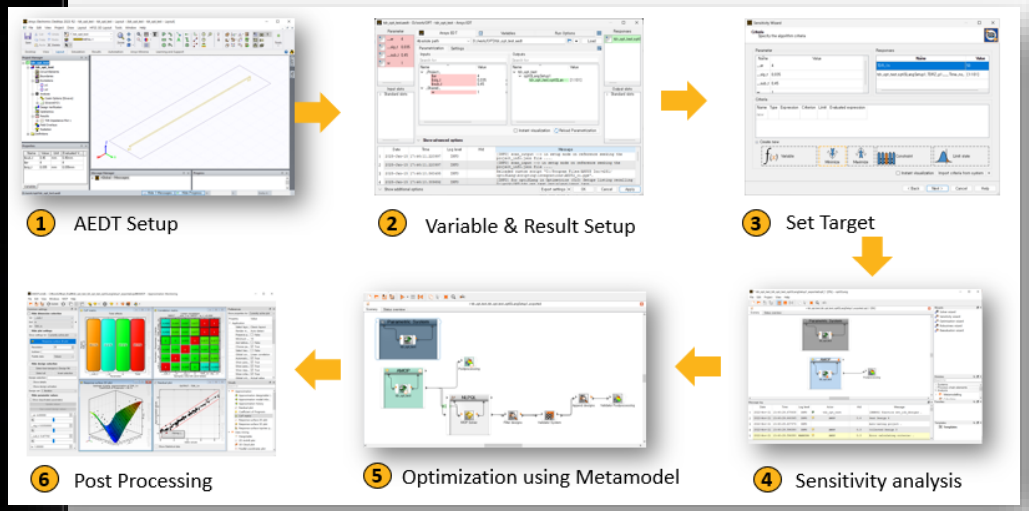
2. Iteration 중에 랜덤하게 tool이 멈추기도 함

- 대부분의 최적화 전문 S/W의 공통 문제점
- 많은 이유로, 반복적인 해석 중에 멈추는 경우가 발생할 수 있음
- System resource 문제와 관련이 깊어서, 성능이 낮은 PC일수록 확률이 높아짐
- 많은 경우 다시 처음부터 시작하면 정상으로 돌기도 함. (어렵고 짜증나는 부분)

Metamodel Automation using PyoptiSLang (25R1)



Progress GUI



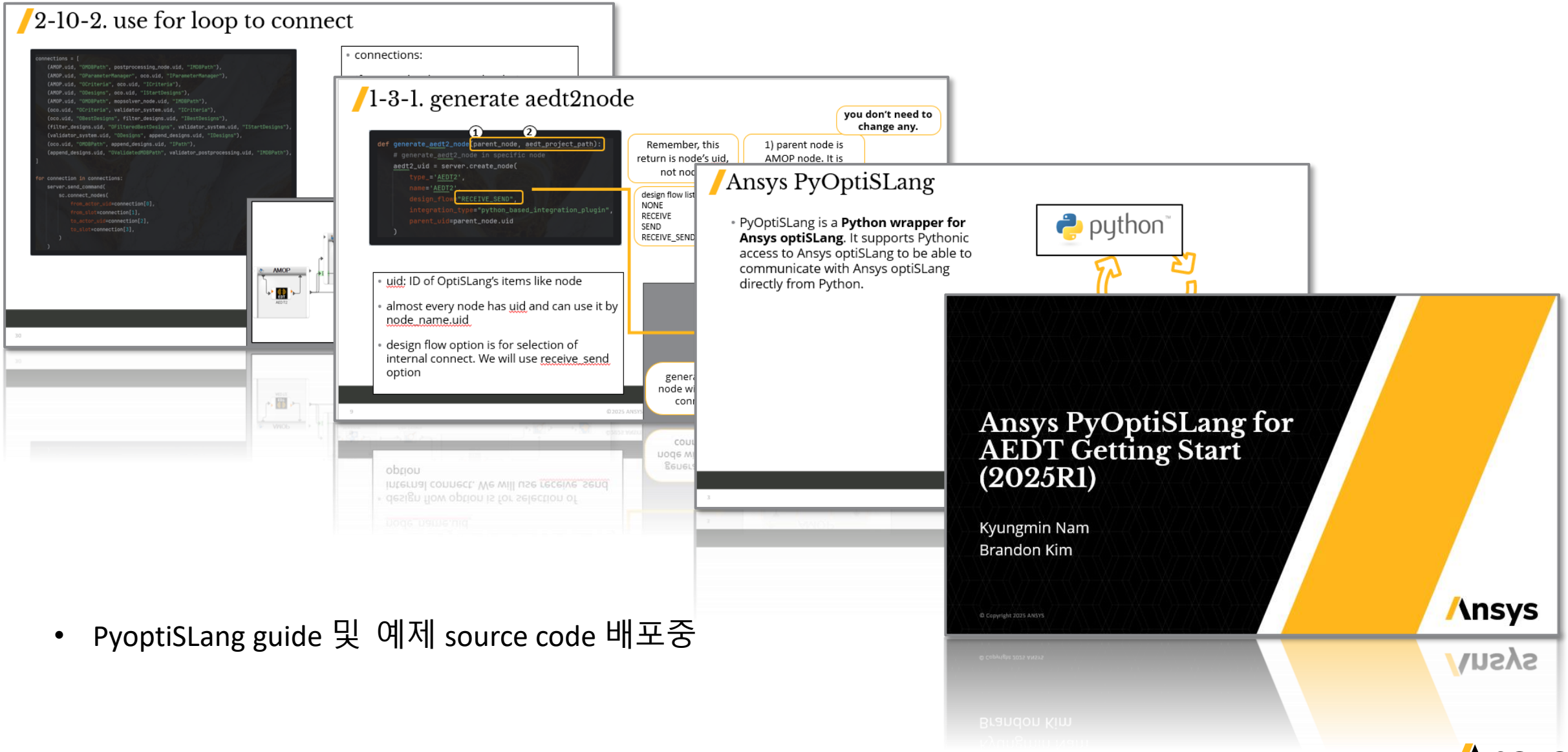
* 실제로는 Non-GUI 로 동작

```python
from ansys.optislang.core import Optislang
from ansys.optislang.core.tcp import server_commands as sc
from ansys.optislang.core.tcp.nodes import TcpProxySolverNodeProxy
from ansys.optislang.core.nodes import IntegrationNode

# ================================================================== #
# ==================  Parameters for Optimization  ================== #
# ================================================================== #

AMOP_len = 10
oco_len = 300
Val_len = 10
aedt_project_path = r"C:\Custom\Optislang\AEDT\tdr_opt_test.aedt"
save_path = r"C:\Custom\Optislang\tdr_opt_test_2.opf"


# ================================================================== #
def generate_aedt2_node(parent_node, aedt_project_path):
    # generate_aedt2_node in specific node
    aedt2_uid = server.create_node(
        type_='AEDT2',
        name='AEDT2',
        design_flow="RECEIVE_SEND",
        integration_type="python_based_integration_plugin",
        parent_uid=parent_node.uid
    )

    # Path info
    json_input_path = {
        "base_path": "",
        "base_path_mode": "ABSOLUTE_PATH",
        "split_path": {"head": "", "tail": str(aedt_project_path)}
    }

    server.send_command(
        sc.set_actor_property(
            actor_uid=aedt2_uid,
            name="Path",
            value={"path": json_input_path}
        )
    )
```

PyoptiSLang Code

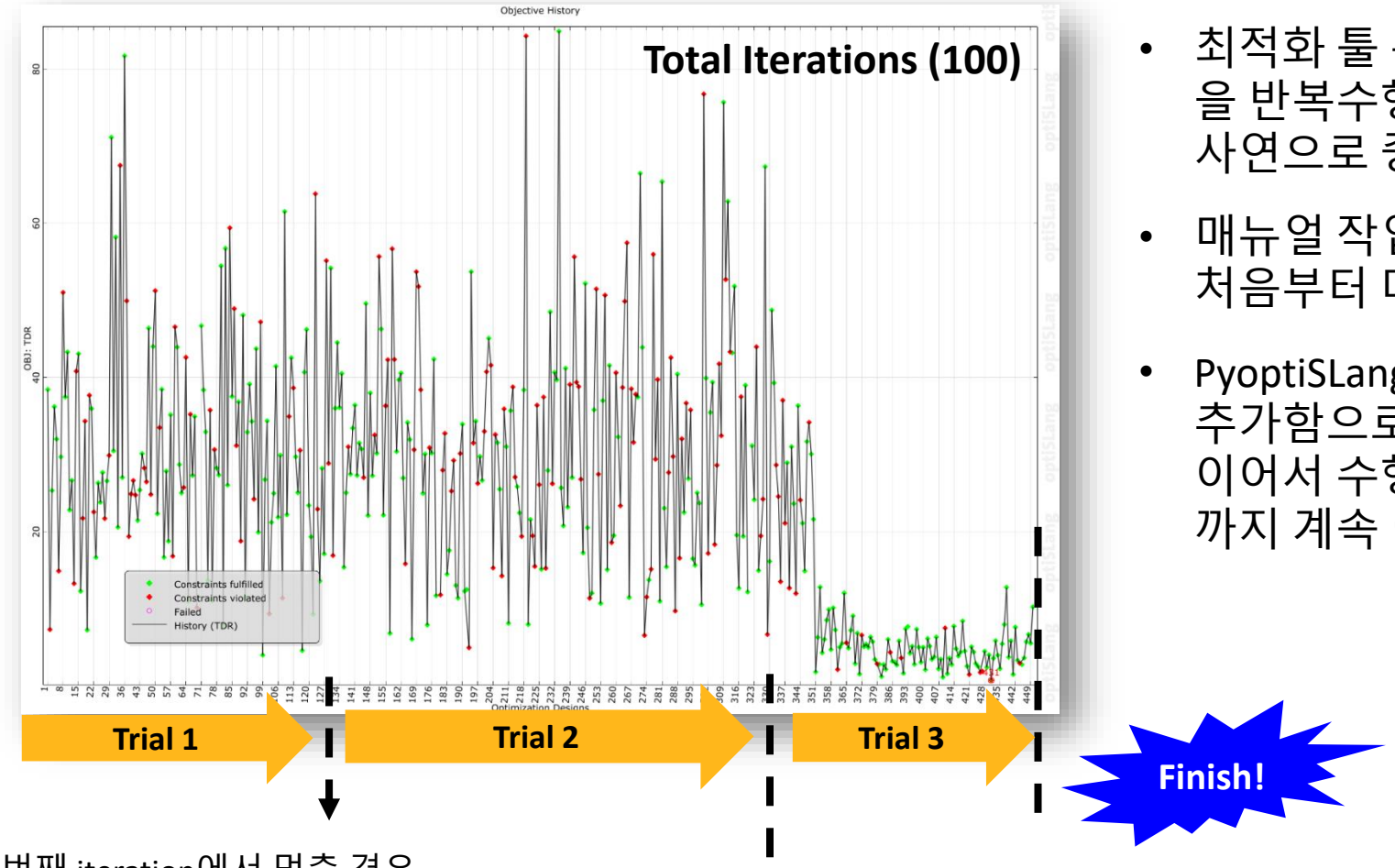
- optiSLang을 이용한 Metamodel 생성의 모든 과정을 PyoptiSLang 으로 자동화 가능 (25R1 버전부터 동작)
- 대중적인 Python 언어로 되어 있기 때문에, 다양한 기능을 coding으로 추가할 수 있음

PyoptiSLang Guide & Example code



- PyoptiSLang guide 및 예제 source code 배포중

Metamodel Automation : Auto Resuming



- 최적화 툴 특성상, 입력 변수 변경에 따른 해석을 반복수행 (iteration) 하다보면, 여러가지 사연으로 중간에 멈추는 경우가 발생
- 매뉴얼 작업 시에는 사람이 상황을 인지하고 처음부터 다시 실행시키는 수밖에 없음
- PyoptiSLang 기반의 자동화에 Python coding을 추가함으로써, 중간에 멈춘 지점에서 자동으로 이어서 수행하게 할 수 있음 (iteration을 끝낼 때까지 계속 이어서 반복시킴)

Conclusion : Metamodel Automation

1. AI는 빅 데이터로 훈련된 추론 엔진.
2. optiSLang을 이용하면 Simulation data로 훈련된 Metamodel 생성 가능
3. optiSLang은 Data filtering 역할도 수행하며, 의미 있는 해석결과만 수집함으로써 데이터 수집 및 해석 효율을 극대화할 수 있음
4. 잘 훈련된 Metamodel에게 목표성능을 제시하면 즉시 설계변수를 도출함으로써, AI와 같은 역할을 수행
5. 이러한 Metamodel 생성과정은 PyoptiSLang을 이용하여 자동화 가능

4. ADRP (AI-Driven Routing Project)

Dream of AI-Driven Routing

- **모든 PCB / Package / Interposer 설계자들의 목표 :**

  → A와 B 지점을 최소한의 공간과 최선의 신호 성능으로 연결

- **그리고 시대의 화두 : 어떻게 하면 AI를 활용하여 빠르게 최적화된 routing을 할 수 있을 것인가?**

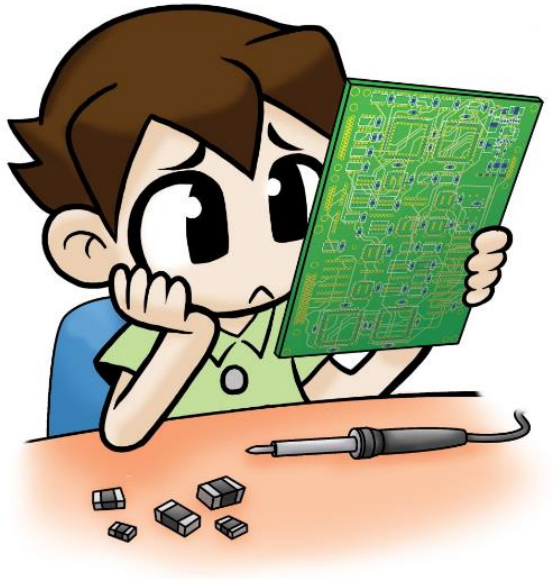


Image drawn by Cartoonist

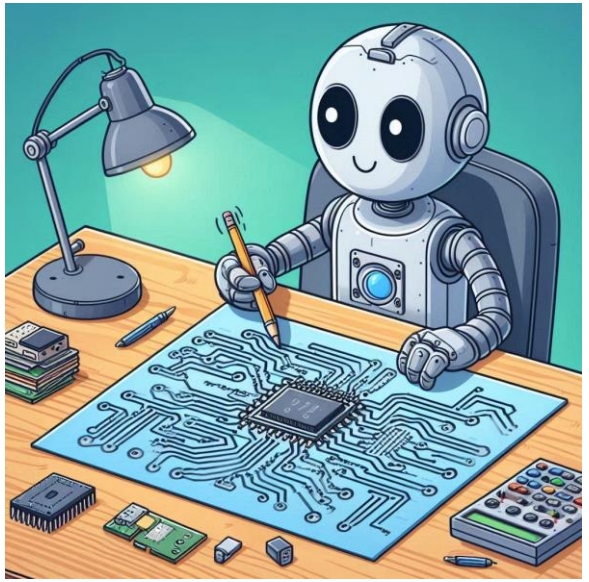
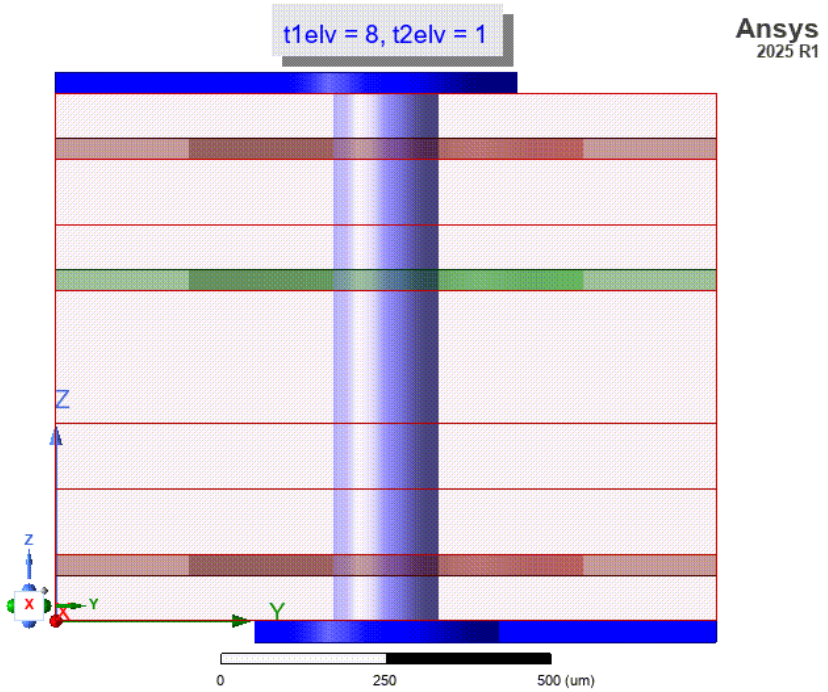


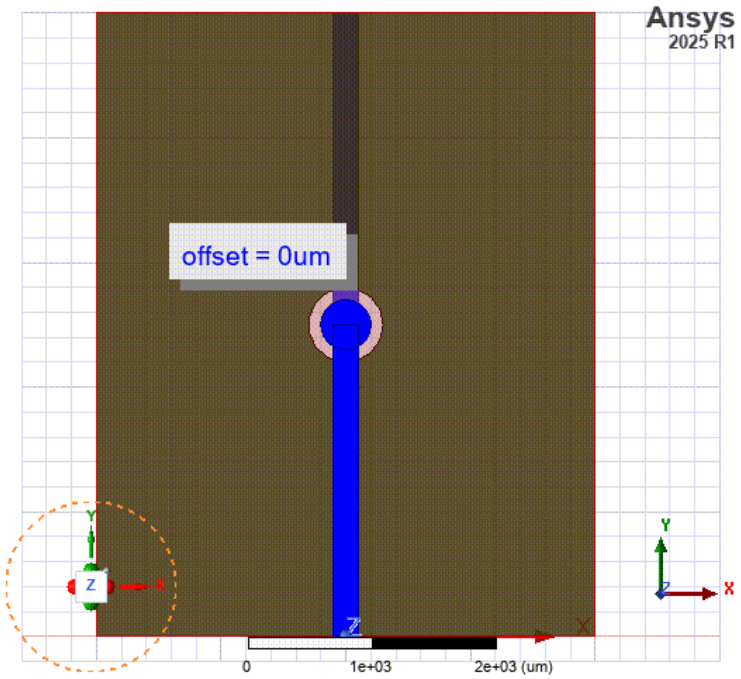
Image generated by AI

Considerations for Auto-Routing



Optimum vertical path

- TDR
- Insertion Loss
- Return Loss

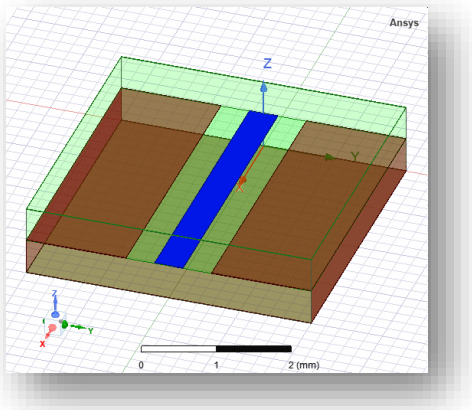


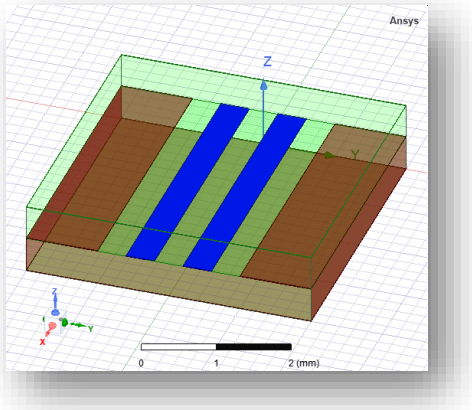
Optimum horizontal path

- Crosstalk
- Skew

** Simple single line example*

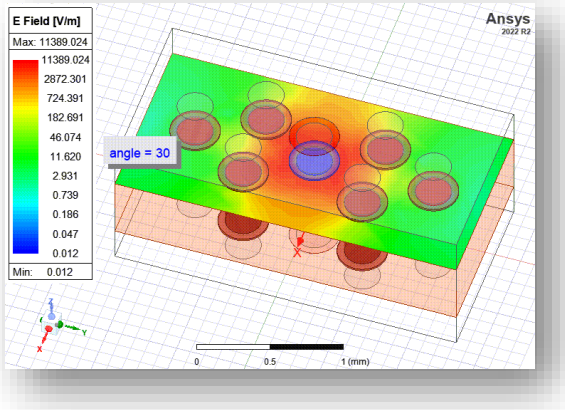
3 Steps : Concept of AI-Driven Routing Project

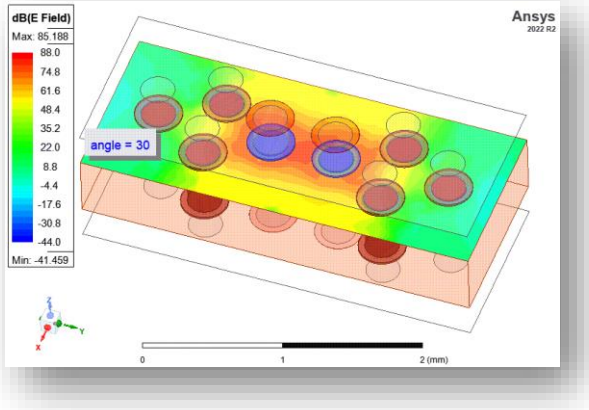




Trace cross section

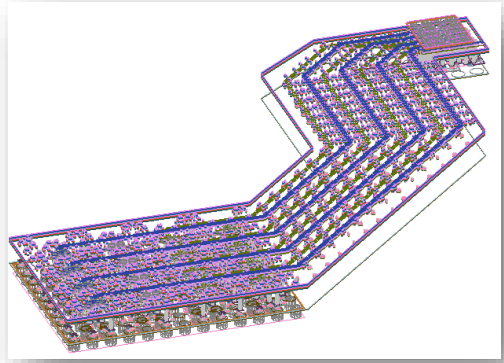
Layer location / Ground condition





Via Connection

Guard via / dimension



Optimum Path

Optimum combination of trace cross section & via connection

Pre-condition for AI-Driven Routing

- **AI 구현의 핵심은 선택과 집중으로 성능을 높이는 점.**

1. 실제 PCB에 사용될 적층정보 입력

- Layer 개수와 높이, 재질 조건은 고정한 상태에서 선로 최적화에 집중
- Siwave의 stk 파일 import 또는 규격화된 format의 excel 또는 CSV 파일 import(customization)

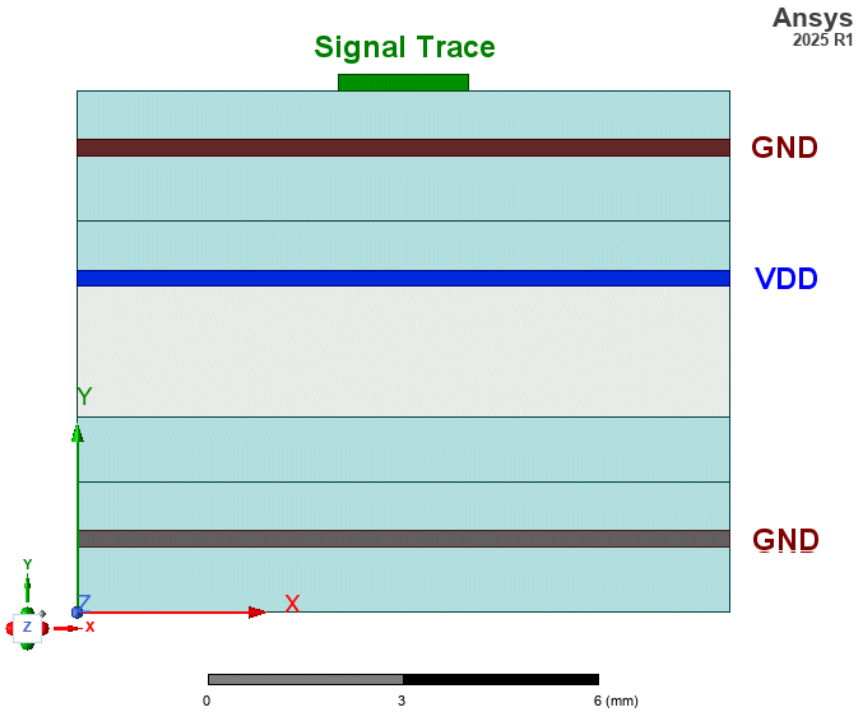
2. Signal 층과 VDD/GND 층은 미리 구분

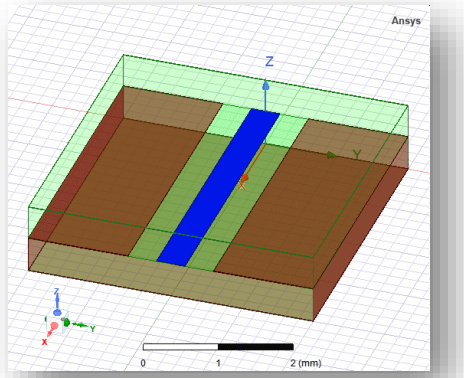
- 전체 Layer에서 signal이 존재 가능한 층을 지정
- VDD/GND Layer는 plane으로 fill 되어 있다는 전제

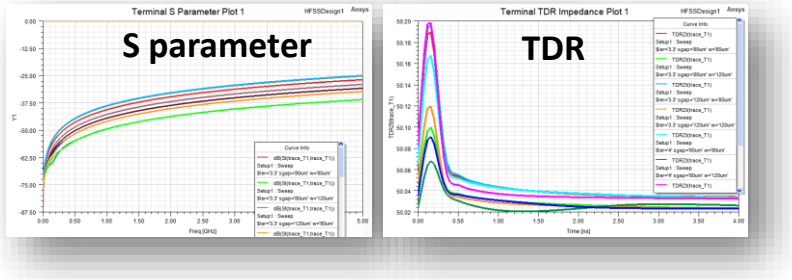
3. Via model 지정

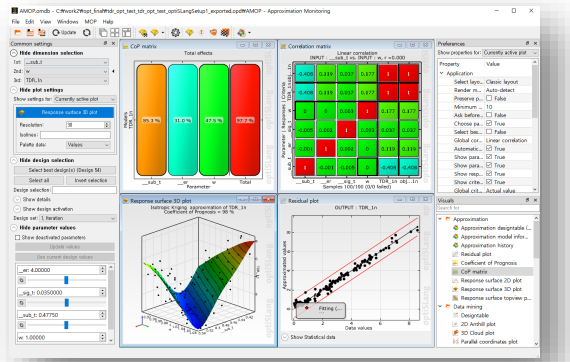
- Through Via 또는 Blind/Buried Via
- Blind/Buried Via 의 경우 가용한 층간 이동수를 지정
- Guard via 유무 지정

Step 1 : Cross-Section Metamodel









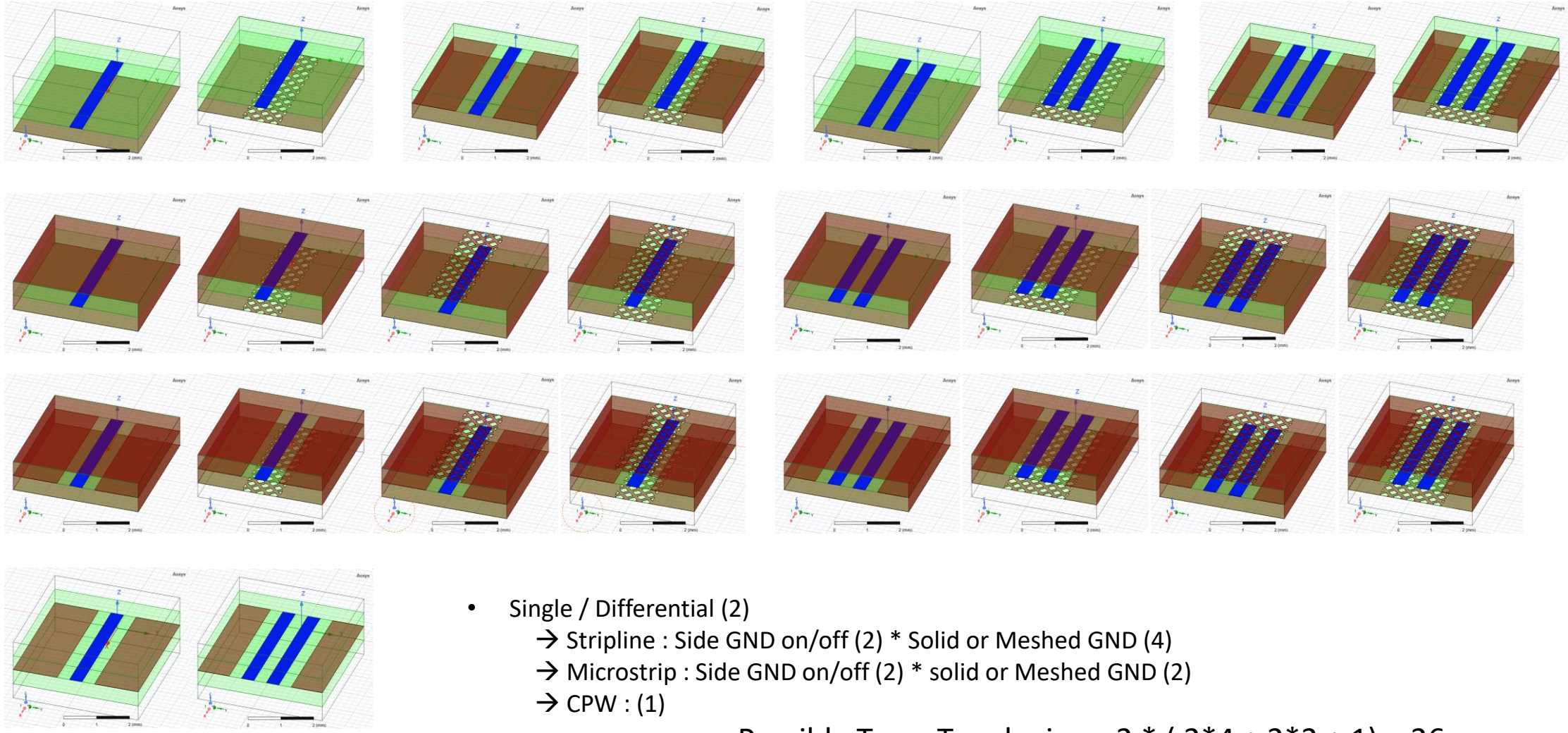
optiSLang

Metamodel Automaton

- 적층 정보 import
- 신호선을 배치할 층과 Power/GND plane층을 미리 지정

- 각 층에 맞는 HFSS 모델 자동 생성
- 변수화 해석을 통해 대량의 훈련용 해석결과 자동추출

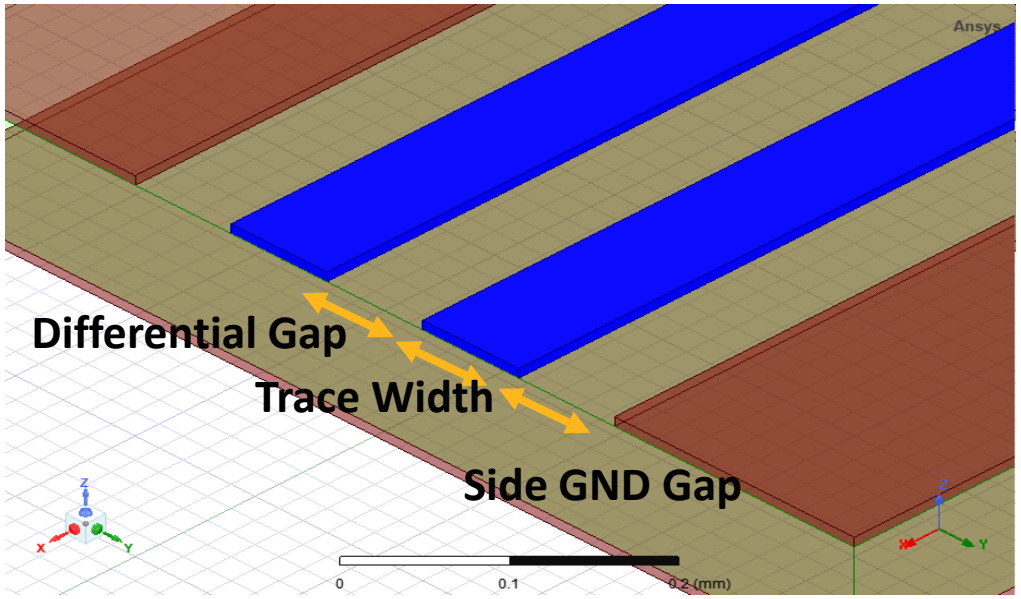
Trace Topologies for Metamodel

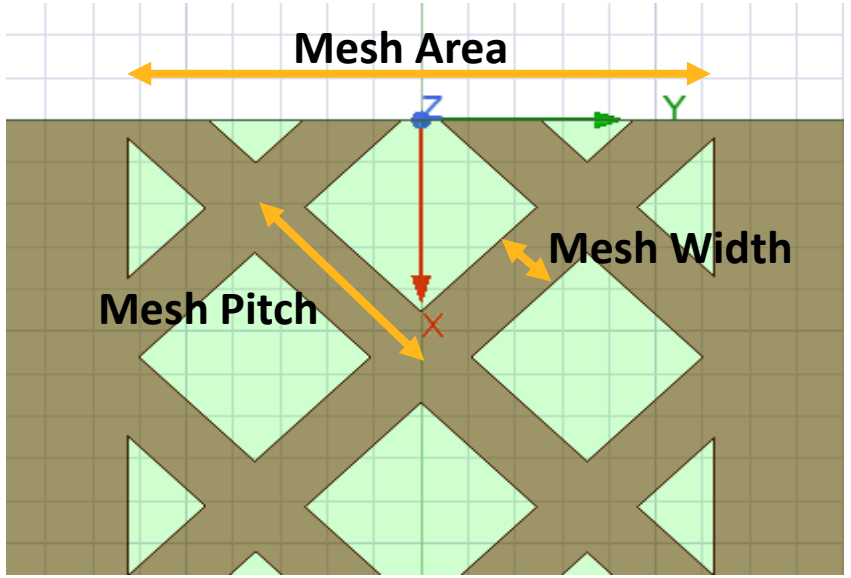


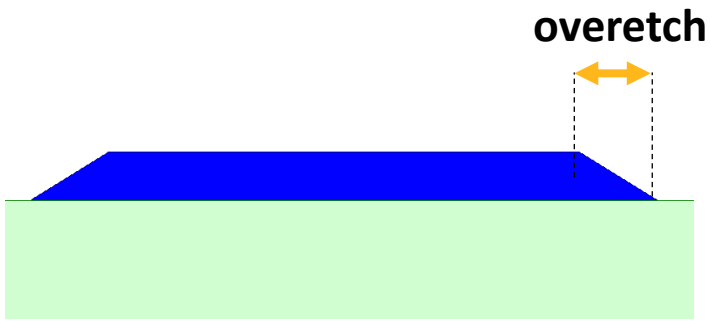
- Single / Differential (2)
  - → Stripline : Side GND on/off (2) * Solid or Meshed GND (4)
  - → Microstrip : Side GND on/off (2) * solid or Meshed GND (2)
  - → CPW : (1)

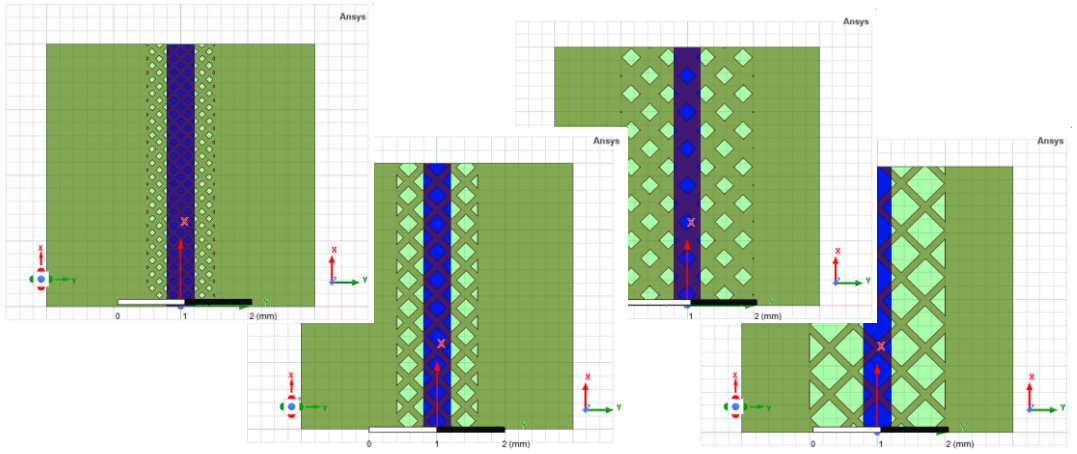
Possible Trace Topologies = 2 * (2*4 + 2*2 + 1) = 26

Trace Design Variables

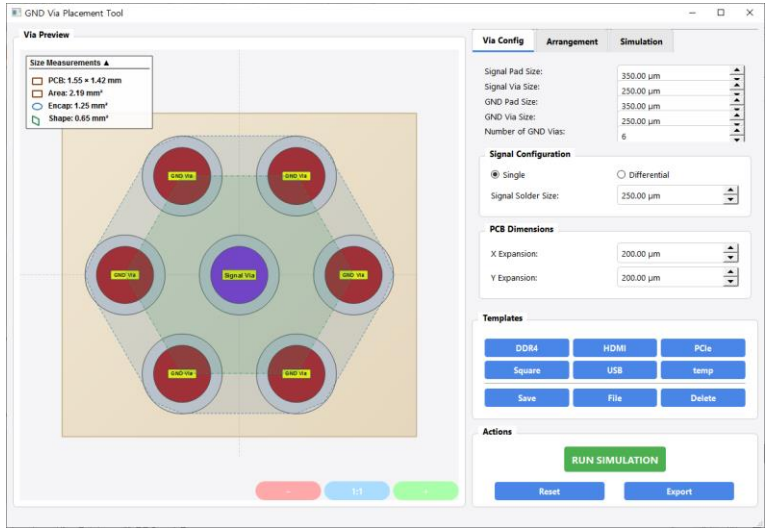




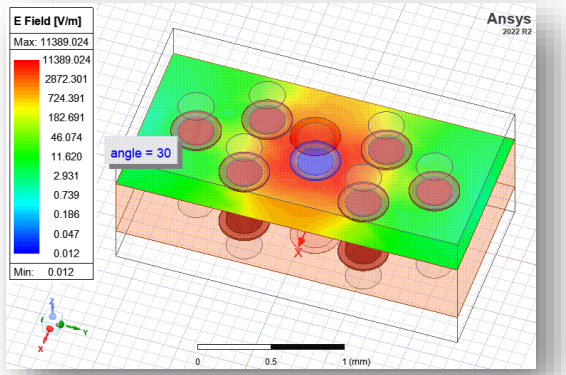


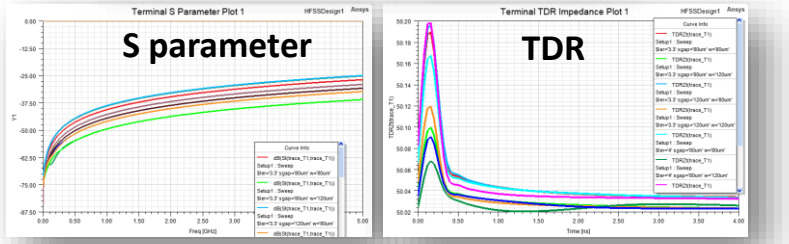


Step 2 : Via Metamodel

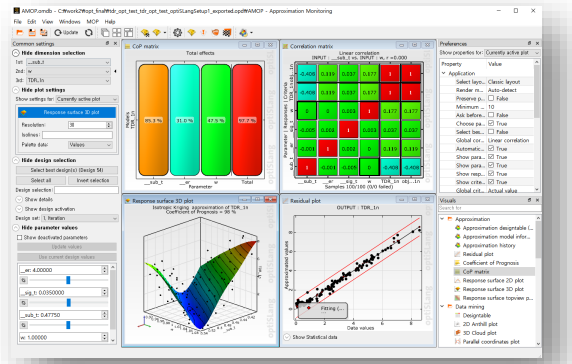


- Via 생성 자동화 code 사용





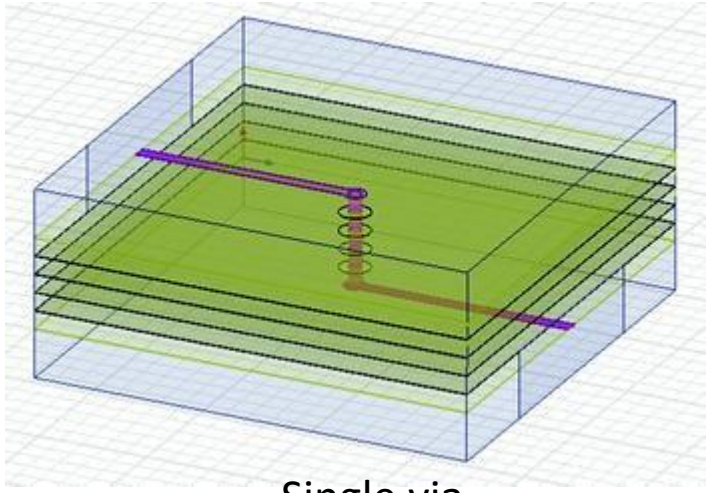
- HFSS Via model 자동 생성
- 변수화 해석을 통해 대량의 훈련용 해석결과 자동추출



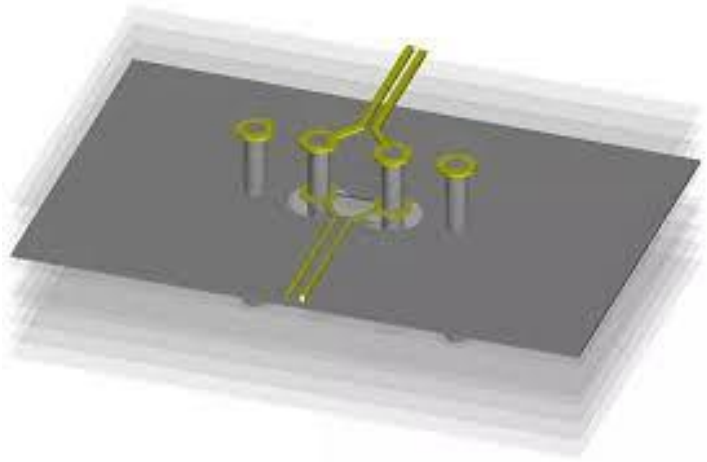
optiSLang

Metamodel Automaton

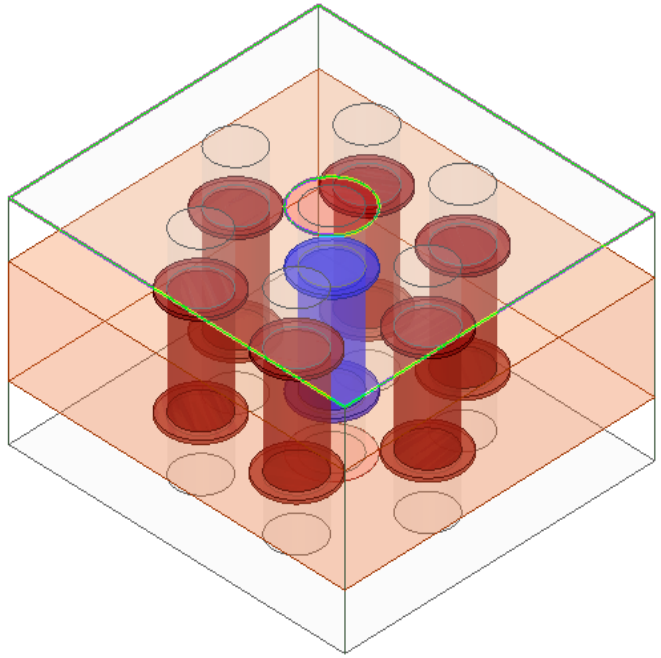
Via Types



Single via



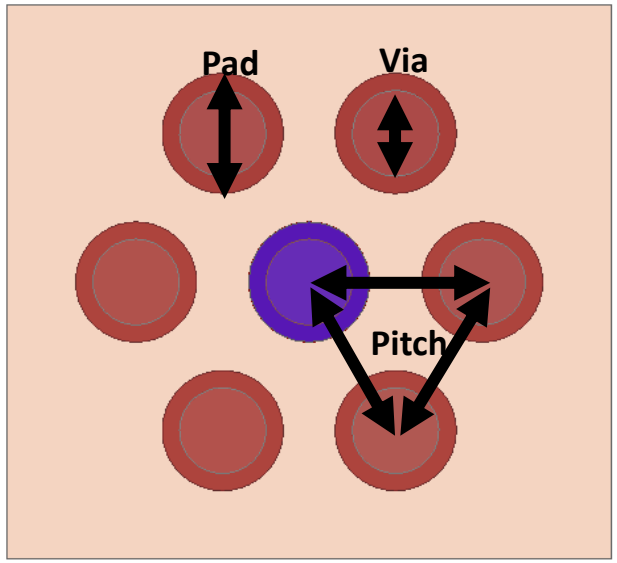
Differential via with gourd via

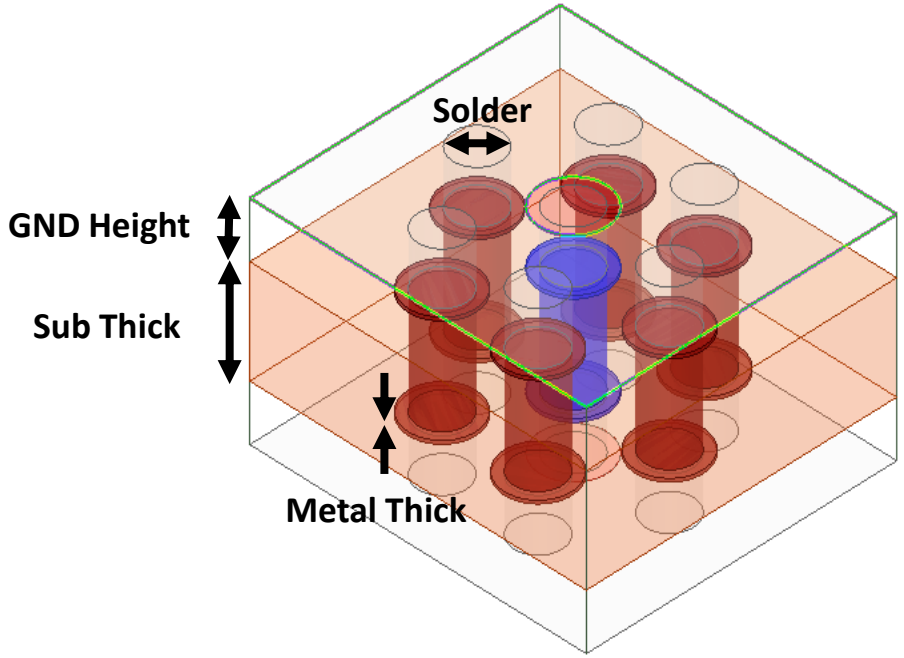


Interposer via with guard via

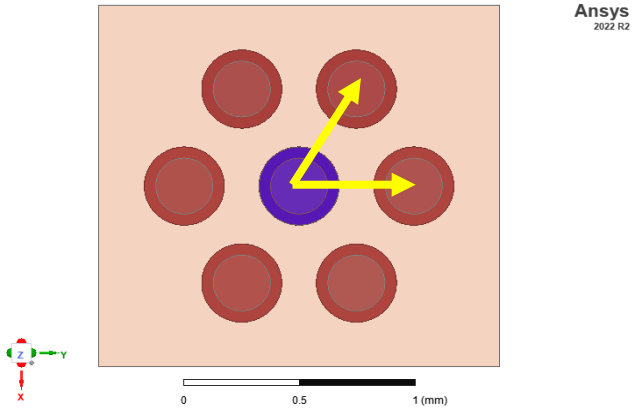
- 기타 custom한 type의 via들이 모델링 자동화에 추가 가능

Example Via Model : Design Variables

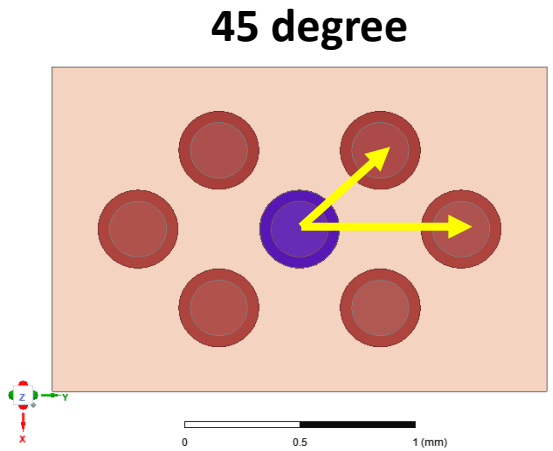




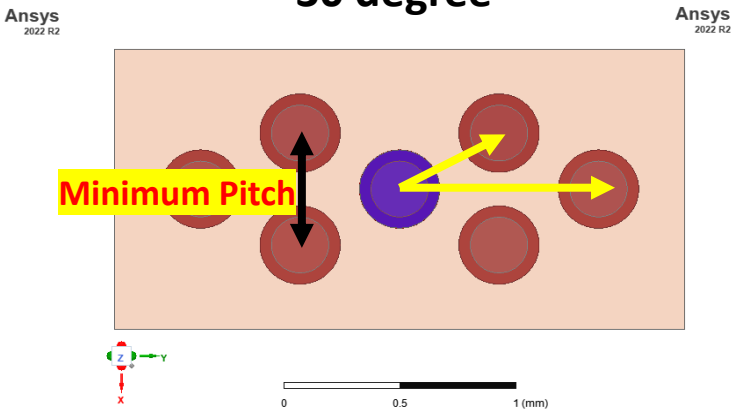
60 degree



45 degree



30 degree



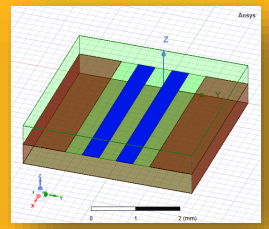
Completed Metamodel

- Target performance
  - TDR
  - S21 / S11

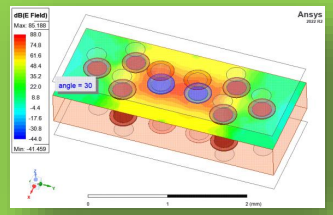
Question

Question

Cross-Section Metamodel



Via Metamodel



Answer

- Best Signal Layer
- Best Trace width

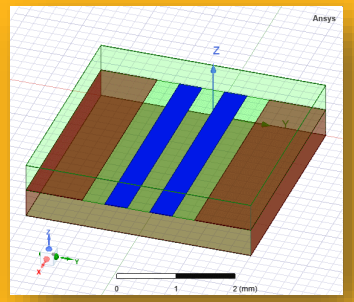
Answer

- Best Via Dimensions
- Best guard via condition
- Smallest area (optional)

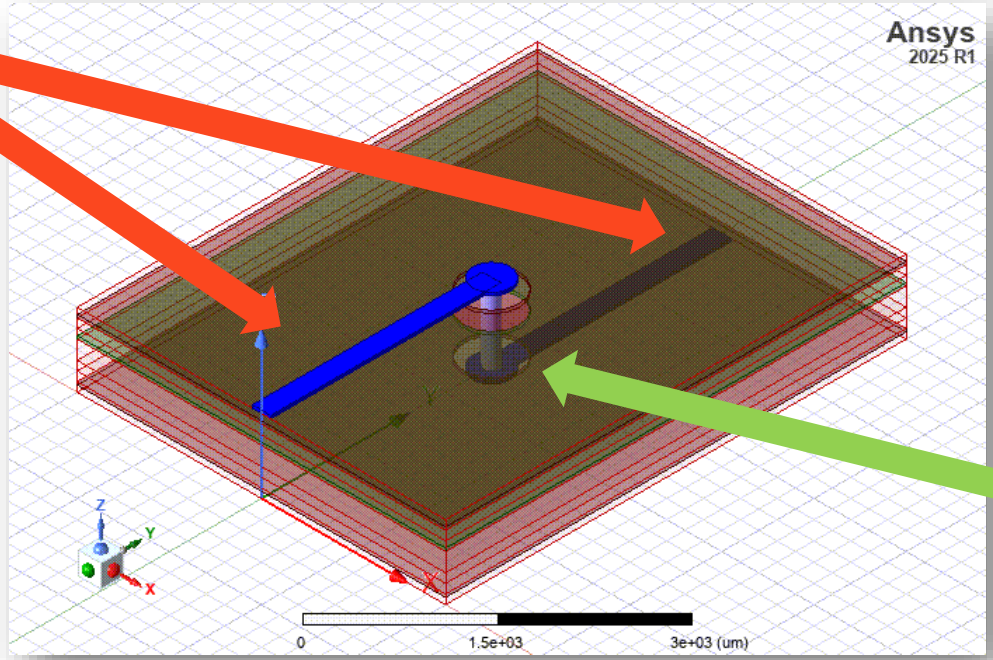
- Metamodel 은 Chat GPT 처럼 질문에 대해 수초 내에 답을 도출할 수 있음

Final Step : Metamodel Combination by Python

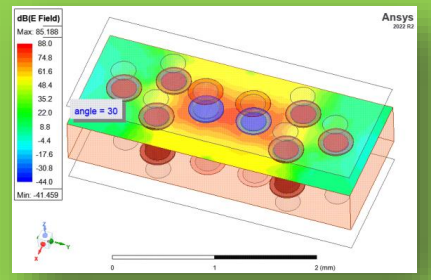
Cross-Section Metamodel



** Simple single line example*



Via Metamodel



- Trace와 via가 반복되는 구간에서 최적의 성능을 내는 trace + via 조합을 빠르게 도출
  (프로젝트 목표 : 고도로 복잡한 적층의 PCB에서도 1분 이내)

Conclusion

- Metamodel을 이용한 PCB routing 자동 최적화에 대한 개념 소개
- 아래와 같은 자동화 프로세스가 1차 완성됨
  - metamodel automation using pyOptislang
  - HFSS model generation for trace/cross-section
  - HFSS model generation for via

- 상황 & 계획

  - 현재 진행형 프로젝트 (~25년말)
  - metamodel combination process에 대한 고민이 더 필요
  - 실제 복잡한 고객사 PCB 모델로 test중
  - gds 기반의 HBM용 silicon interposer 적용을 검토/분석중
  - 다양한 형태의 via model 자동화 추가 예정 (고객 요청여부에 따라)