Multi Chip Analysis & Debug Using RedHawk-SC

RedHawk-SC Modularized Training Series 2021_R1

Ansys

Info For Attendees

- Please join audio via Audio Broadcast option
- Please use the Q&A window to clear queries and one of the panelists will answer it.
  - Direct questions to all of panelists
- This training is for 2 hours with 10 mins Q&A session at the end
- The slides and recording will be available at Ansys website within a week
  - Registered participants will be receiving emails with the link
- For offline follow up of queries, please reach out to your local AE or email anudeep.surasani@ansys.com

People on Panel

- **Host**
  - Abhijith M V – Lead Application Engineer
  - Anudeep Surasani – Lead Product Specialist
- **Panelists**
  - Greeshma Prakasan – Senior Product Specialist
  - Prajwal Holla – Product Specialist

Prerequisites for the training

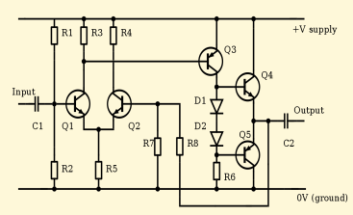
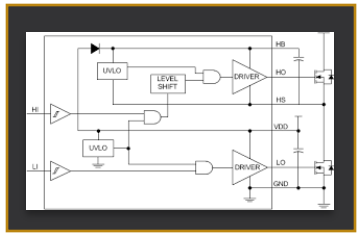
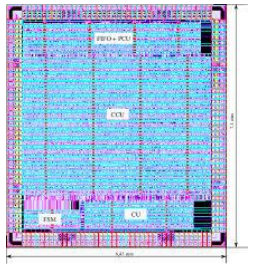
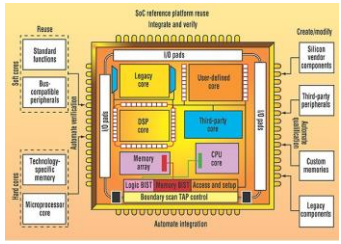
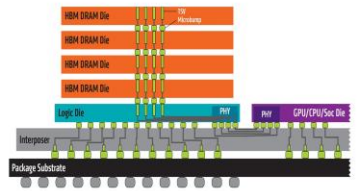
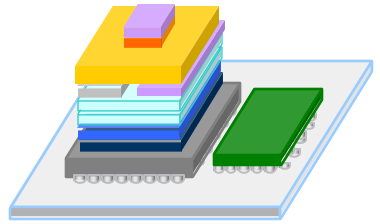
No	Training Program	Expectations – Must Know
1	Chapter 01 : Introduction_to_SeaScape	• Reading in input data, performing data integrity checks and creating base views
2	Chapter 03: Static IR / Power EM Analysis & Debug	• Basics of Static & Power EM Analysis
3	Chapter 04: Vectorless Dynamic Analysis in RedHawk-SC	• Basics of Dynamic Analysis

Training Agenda

- Introduction to 3DIC
- Multi Chip Flow Overview
- Multi Chip Specific Input Data Requirements
- Multi Chip Flow Setup
  - Config Creation
  - Command File Creation
- Different Multi Chip Analysis flows
  - Full Detailed flow
  - On-the-fly Model based flow
  - Model Based flow
  - Interposer PDN Quality Checks
- Training Labs
  - Overview of training Lab
  - Multi Chip specific GUI
  - Reporting Utilities

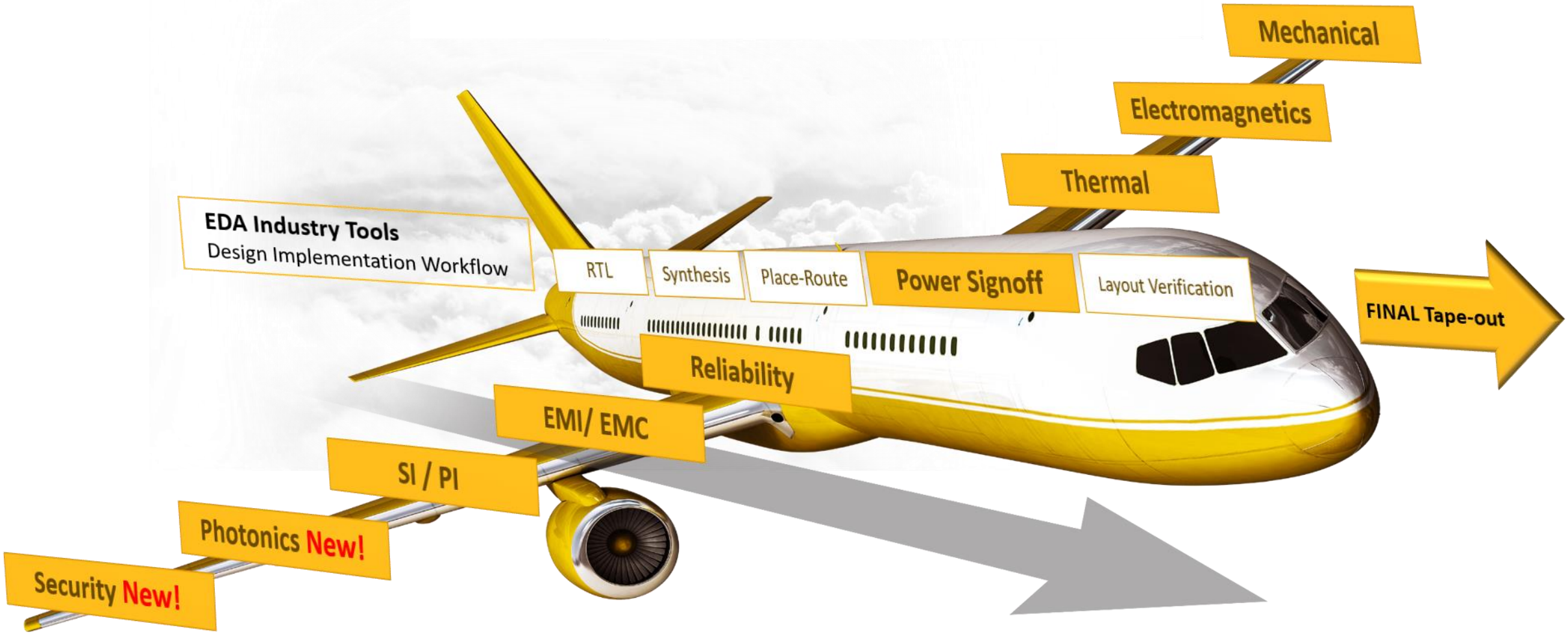
Introduction to 3DIC

Evolution of Design Complexity

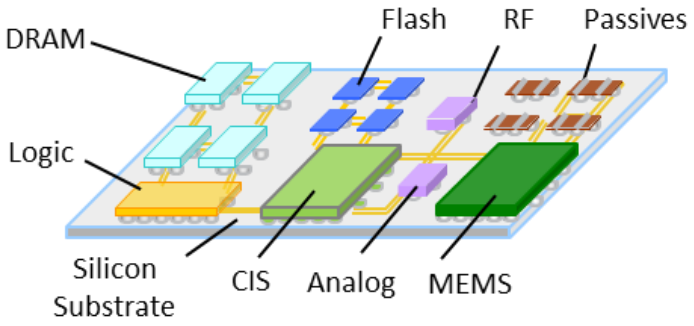
1980	1990	2000	2010	2015	2020
1-2µ	Sub-micron	Deep submicron	Nanometer	Sub 100nm	5/3 nm
Transistor-level	Gate-level	ASIC	System on Chip	Stacked Die	Complex 3DIC
					
Functional	Functional Timing	Functional Timing Power	Functional Timing Power Reliability	Functional Timing Power Reliability Thermal EMI	Functional Timing Power Reliability Thermal EMI Mechanical

References: Various, Applied Materials, Intel, AMD, Google images

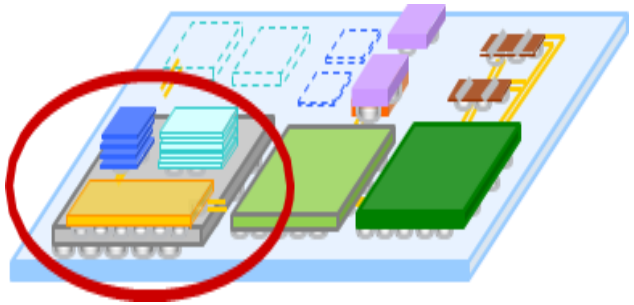
Ansys Multiphysics Simulation Signoff



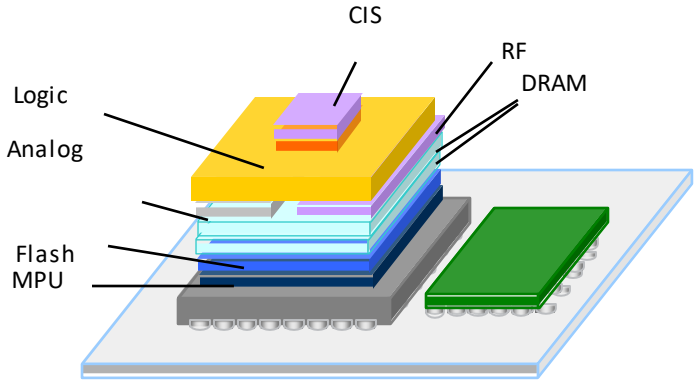
Why we need 3DIC ?



Conventional 2D



2.5D (Silicon Interposer)

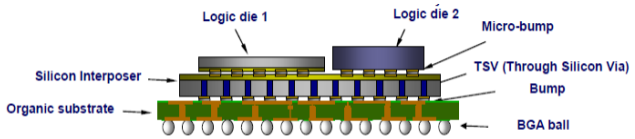


3D Stacked IC

Pros	Cons
• Can use different process nodes different components • Small form factor • Short channel length & propagation delay • Lower power, higher bandwidth • High system performance • Better yield	• Small heat sink area => Heat accumulation • Design implementation is hard • High complexity • Difficult to analyze, verify and signoff • Noise Coupling • Mechanical Stress

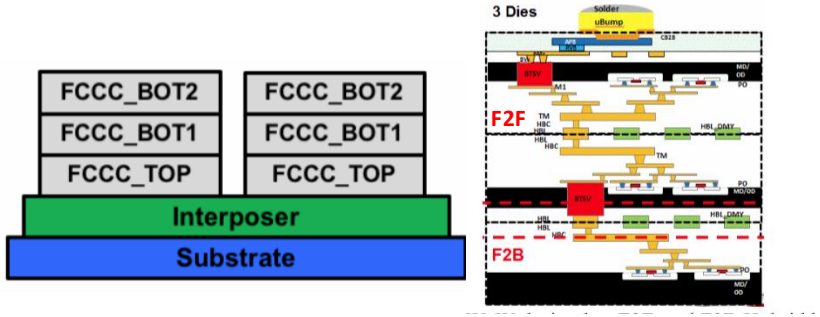
Comprehensive chip, package and system co-design and co-analysis is necessary

Various Multi-Die Structures for reference



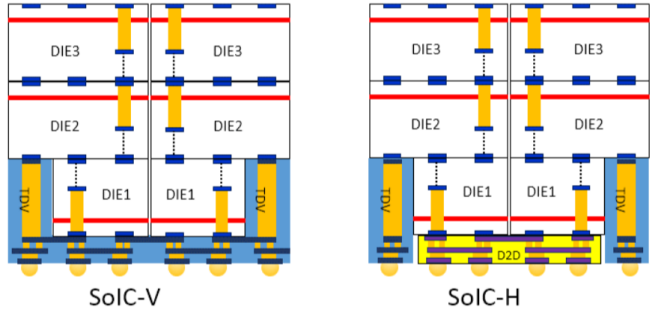
Chip **o**n **W**afer **o**n **S**ubstrate (**CoWoS**) / **2.5DIC**

Advanced packaging technology integrates logic computing and memory chips in a 3-D way.



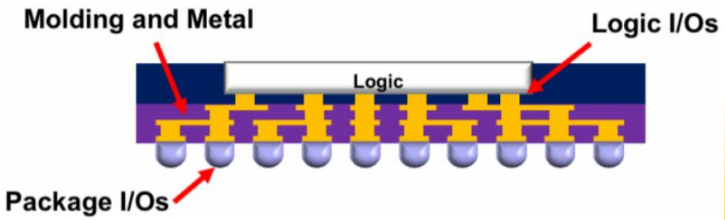
Wafer **o**n **W**afer (**WoW**) (2DIE/3DIE)

*It is multi-die stacking process. Using this Hybrid bonding and TSV, the design can be compact for complex designs. WoW design has 2-die, face-to-face stacking through HBL connection. WoW designs can have **F2F and F2B Hybrid bonding** connection for 3-die stacking.*

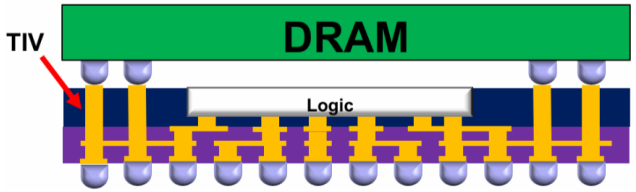


System **o**n **I**ntegrated **C**hips(**SoIC**)

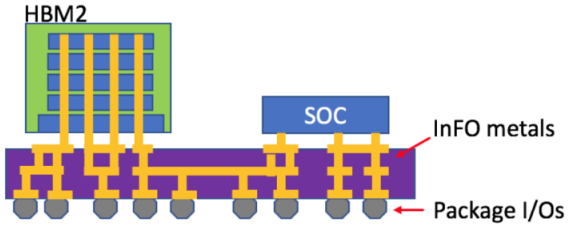
*technology for multi-die stacking using TSV and Hybrid Bumping, also using **TDV(Through Dielectric VIA).** TDV is new package technology. The TDV via can be used for through dielectric or molding.*



Integrated **F**an **O**ut (InFO)



InFO-Package **o**n **P**ackage (**PoP**)

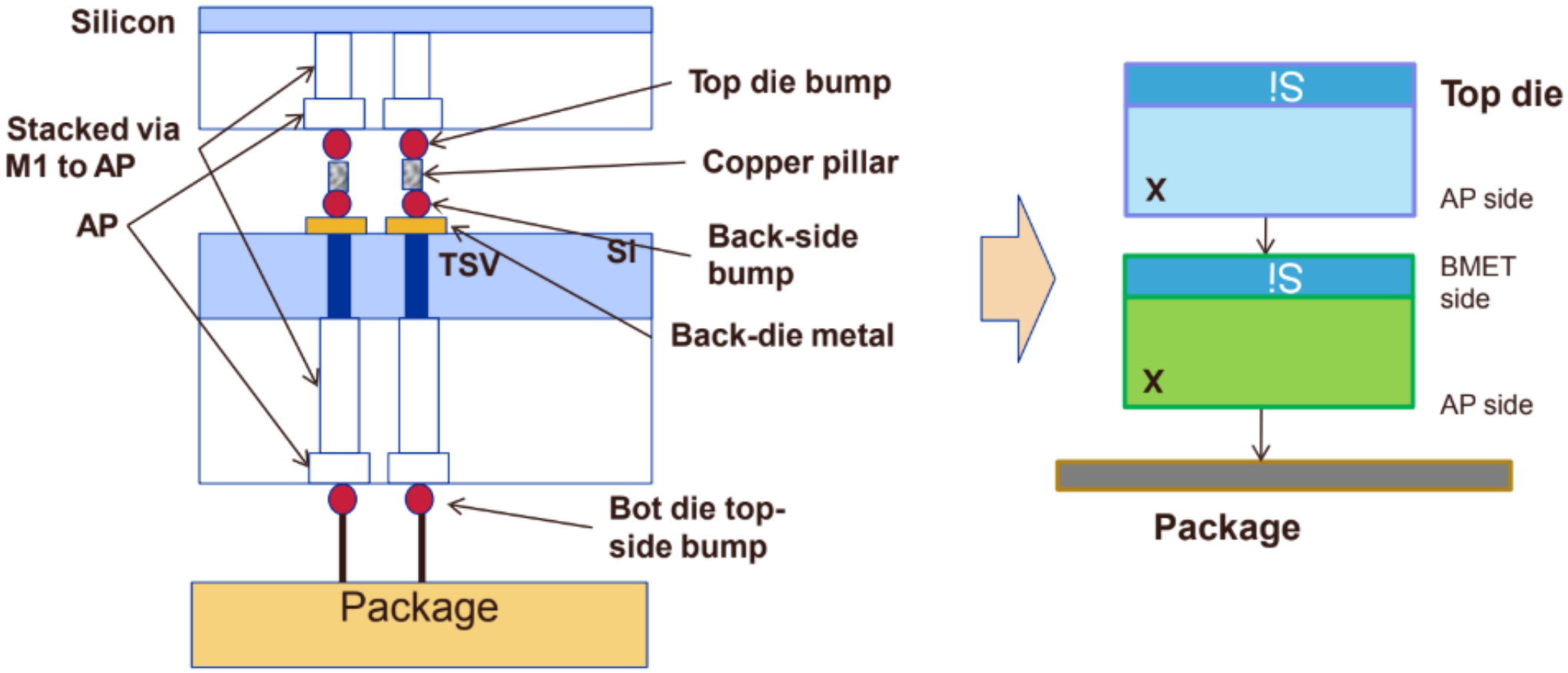


InFO-**M**emory **S**ystem (**MS**)

Integrated Fan Out (InFO) technology, based on wafer molding and fine pitch (5/5um) metal process without substrate, enables reduced thickness, optimized performance, and lower cost for mobile computing products.

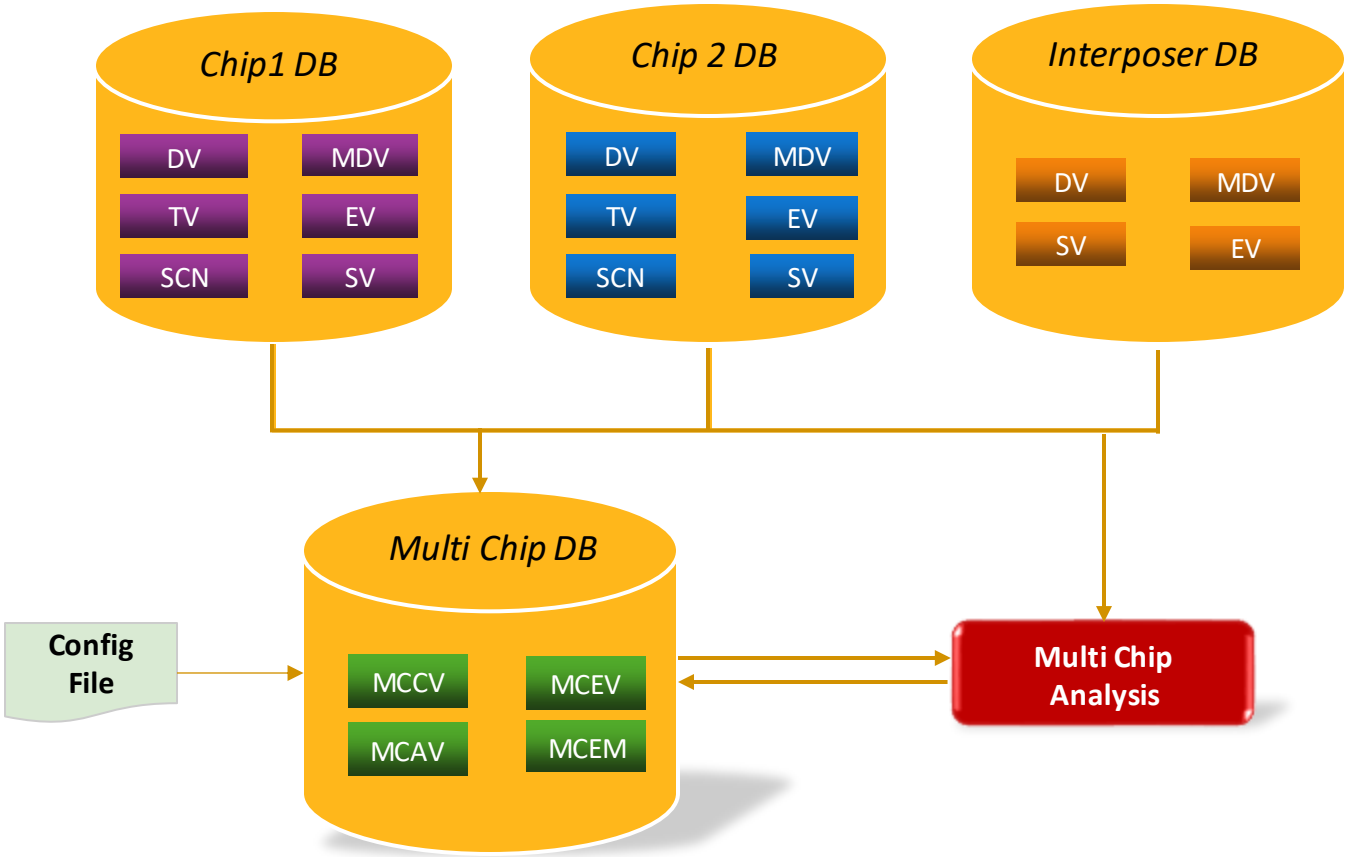
Platform of SoC and HBM (High-Bandwidth Memory) on package substrate, based on wafer molding and fine pitch (2/2 um) metal process, enables reduced thickness, optimized performance, and complex routing products.

What is a TSV ?



Multichip Flow Overview

Multichip Flow Overview



Multi Chip Specific Input Data Requirements

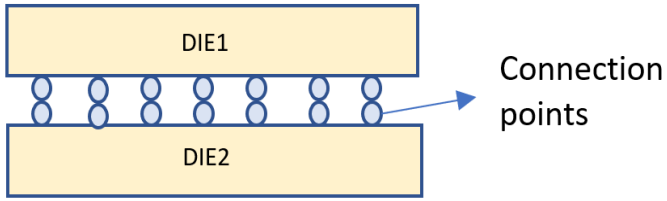
Additional Input Data Requirements

- Required data for RHSC multichip analysis is slightly different from a single chip analysis. Following are the multichip specific design data required.
  - **Connection point location for each chip:** Similar to ploc file, connection point locations for every interfaces are required

```
#Connection_Point_Name      x     y     metal_layer net_name
 frontBumpAP_6_7            1750  1550       AP          VDD
 frontBumpAP_6_14           3150  1550       AP          VDD
 frontBumpAP_6_16           3550  1550       AP          VDD
 frontBumpAP_7_5            1350  1750       AP          VDD
```

  - **Interconnect parasitic for connection points:**
    - The structures which establish connections across different chips.
    - There can be power grid connections as well as signal net connections

```
R  1e-4
L  1e-12
C  1e-15
LENGTH 10
WIDTH 10
HEIGHT 5
```



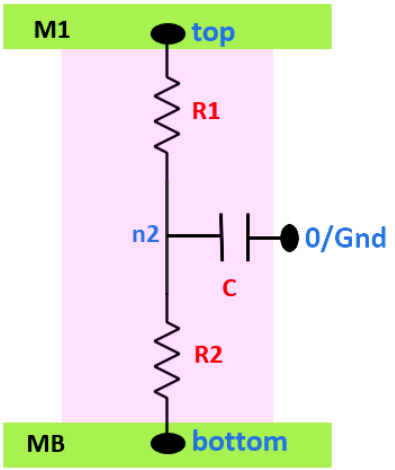
Additional Design Data Requirements

- **TSV Subckt (optional) :**

  There are two ways to derive the TSV parasitics in multi-chip flow:

  - Using Foundry Tech-file (Eg: TSMC) : The tool will extract TSV based on tech file data
  - Using TSV sub-circuit

```
.subckt tsv_rc   top      bottom
        R1       top      n2       0.012
        R2       n2       bottom   0.012
        C        n2       0        1.548E-13
.ends
```



Syntax for parsing tsv sub-circuit to ev:

```
create_extract_view(..., tsv_subckts=tsv_subckts,...)
```

TSV subcircuits definition:

```
tsv_subckts = [
{'layer_name' : '<layer1>', 'subckt_name' : '<subckt_name>','subckt_file' :'<path_to_subckt_file>'},
{'layer_name' : '<layer1>', 'subckt_name' : '<subckt_name>','subckt_file' : '<path_to_subckt_file>'},
.... ]
```

Example:

```
tsv_subckts = [
{'layer_name' : 'TDV', 'subckt_name' : 'tdv_rlc','subckt_file' : './tsv_rlc.subckt'},
{'layer_name' : 'TSV', 'subckt_name' : 'tsv_rlc','subckt_file' : './tsv_rlc.subckt'},]
```

Multichip Config File Creation

What is in Multichip Config file ?

Individual chip Information

- Chip size
- Def Name and Chips Instantiation Names
- Interface definitions
- Position, layer, net, name

Connection points details

- Dimensions and parasitic of the connection points

Multi Chip System

- Connections between instances with locations
- Individual connection point pairs
- Electrical model for connection points

System interface

- Defines the package interface

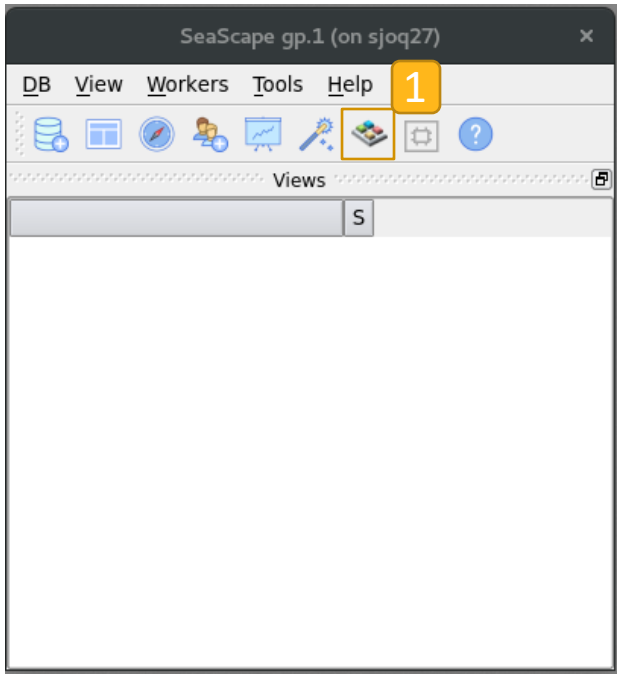
Sample Config File

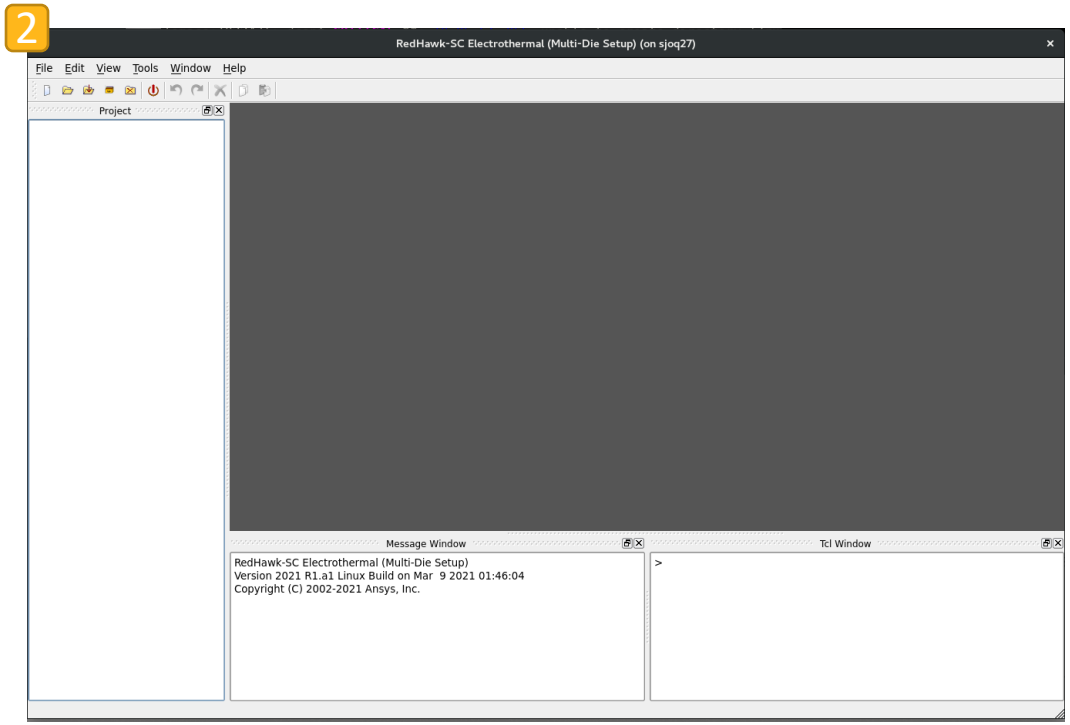
```
DIE Galaxy 1 {                  → Original DIE name from def
        INSTANCES DIE1 DIE2     → Instantiations
        NET  {                  → Nets to be analysed
                VDD POWER IN
                VSS GROUND IN
        }
        INTERFACE IF_DIE_INT {  → All the interface locations
                VDD_DIE_1 28 1220 metal12 VDD
                …                       …
        }
SIZE 0 0 1226.83 1226.4
        SHRINK 1
}
DIE GalaxyInterposer 2 {
        INSTANCES Interposer
        NET  {
                VDD POWER INOUT
                VSS GROUND INOUT
        }

        INTERFACE IF_INT_DIE1 {    → Interfaces to die1
                …
        }

        INTERFACE IF_INT_DIE2 {    → Interfaces to die2
                VDD_INT2_1 2624 1220 UBM VDD
                …
        }
        ….
        INTERFACE IF_INT_C4 {      → Interfaces to C4 (pkg)
                VDD_PKG_1 146 1216 UBMB VDD VDD_VRM
        }
```

```
CONNECTION Interposer_DIE1 {            → Interfaces to die2
        REF_DIE Interposer IF_INT_DIE1 → DIE taken for reference
        CONNECT_DIE DIE1 IF_DIE_INT
        PLACE 21.4292 -0.616852         → Relative placement information
        ORIENT N → Relative Orientation
        CONNECTION_POINTS  {            → Connection between interfaces
                VDD_INT_1 VDD_DIE_1 PGCP → the connection parasitic
type
                …
        }

CONNECTION Interposer_DIE2 {    → connection information to DIE2
        REF_DIE Interposer IF_INT_DIE2
        CONNECT_DIE DIE2 IF_DIE_INT
        PLACE 1446 -6.10352e-05
        ORIENT FN
        CONNECTION_POINTS  {
                VDD_INT2_1 VDD_DIE_1 PGCP
                …
        }

CP PGCP {               → Connection Point parasitic & dimensions
        R 0.001
        L 2e-11
        C 5e-12
        LENGTH 0.8
        WIDTH 0.8
        HEIGHT 0.001
}

SYSTEM  {
        PACKAGE_CONN  { → interface connecting to package
                PLOC_INTERFACE Interposer IF_INT_C4
        }
}
```

Multichip Config creation using setup utility

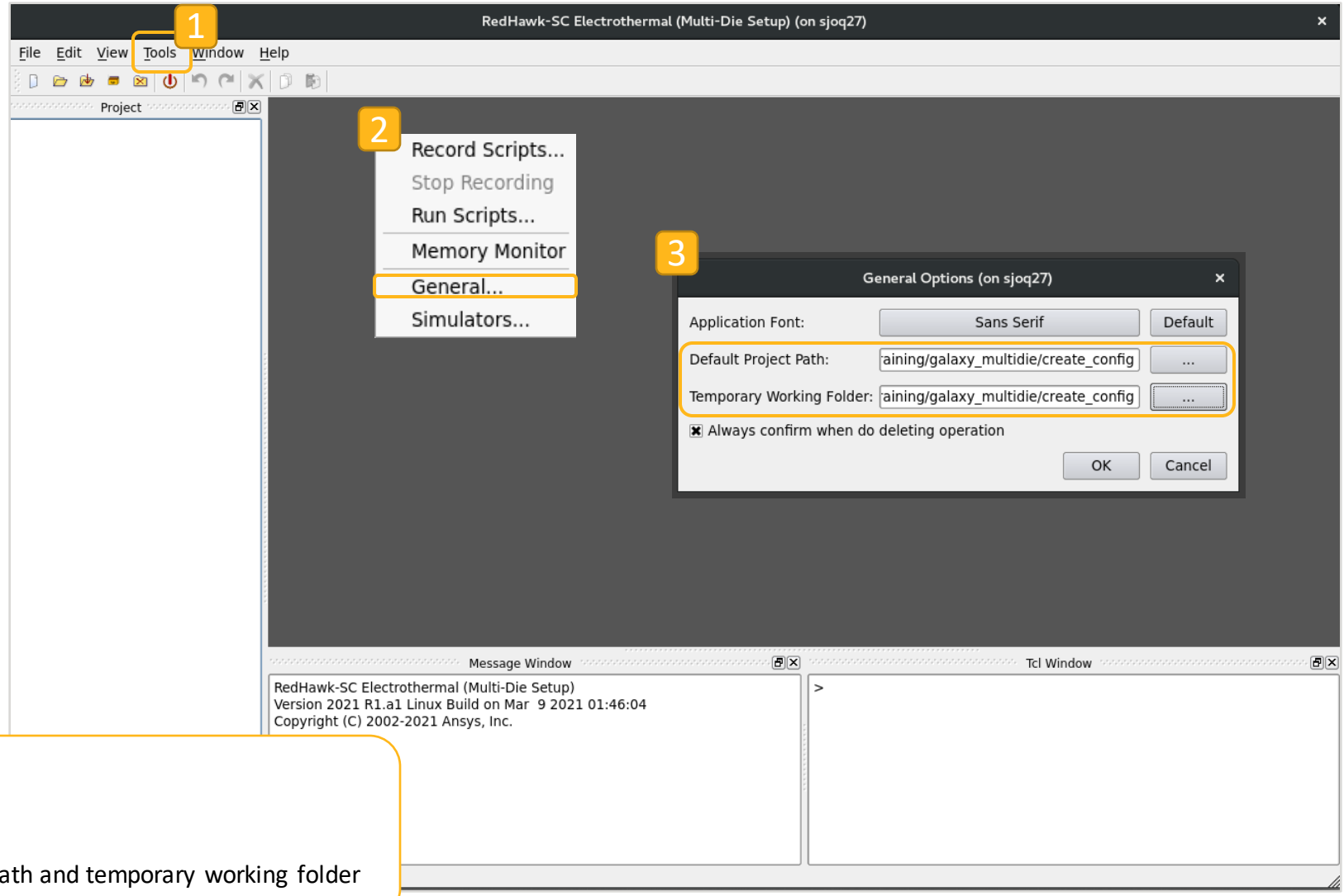
- Launch RedHawk-SC console `redhawk_sc --console`





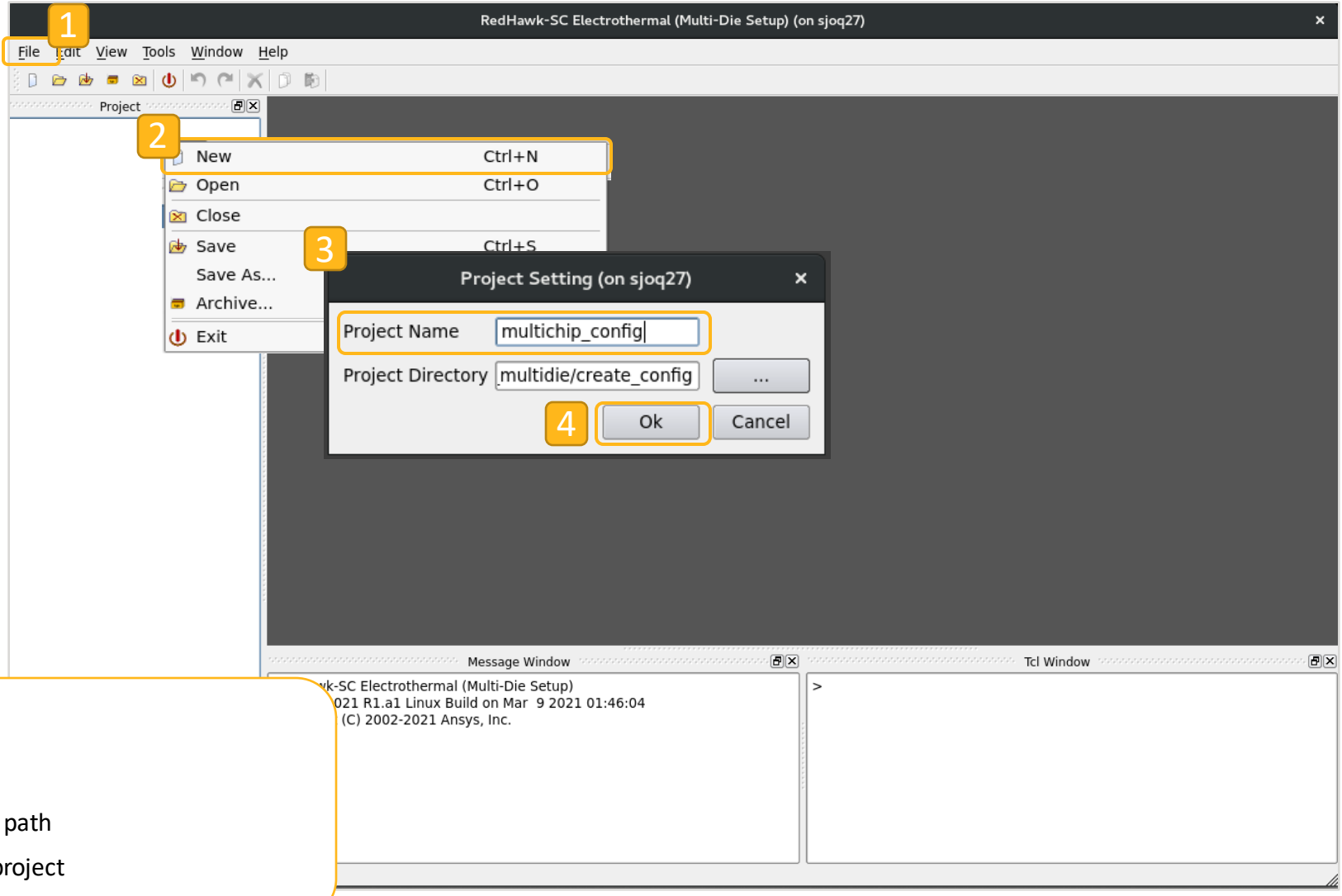
1 Click on Multi-Die Setup Icon
2 The Multi Die Setup Window opens

General Settings



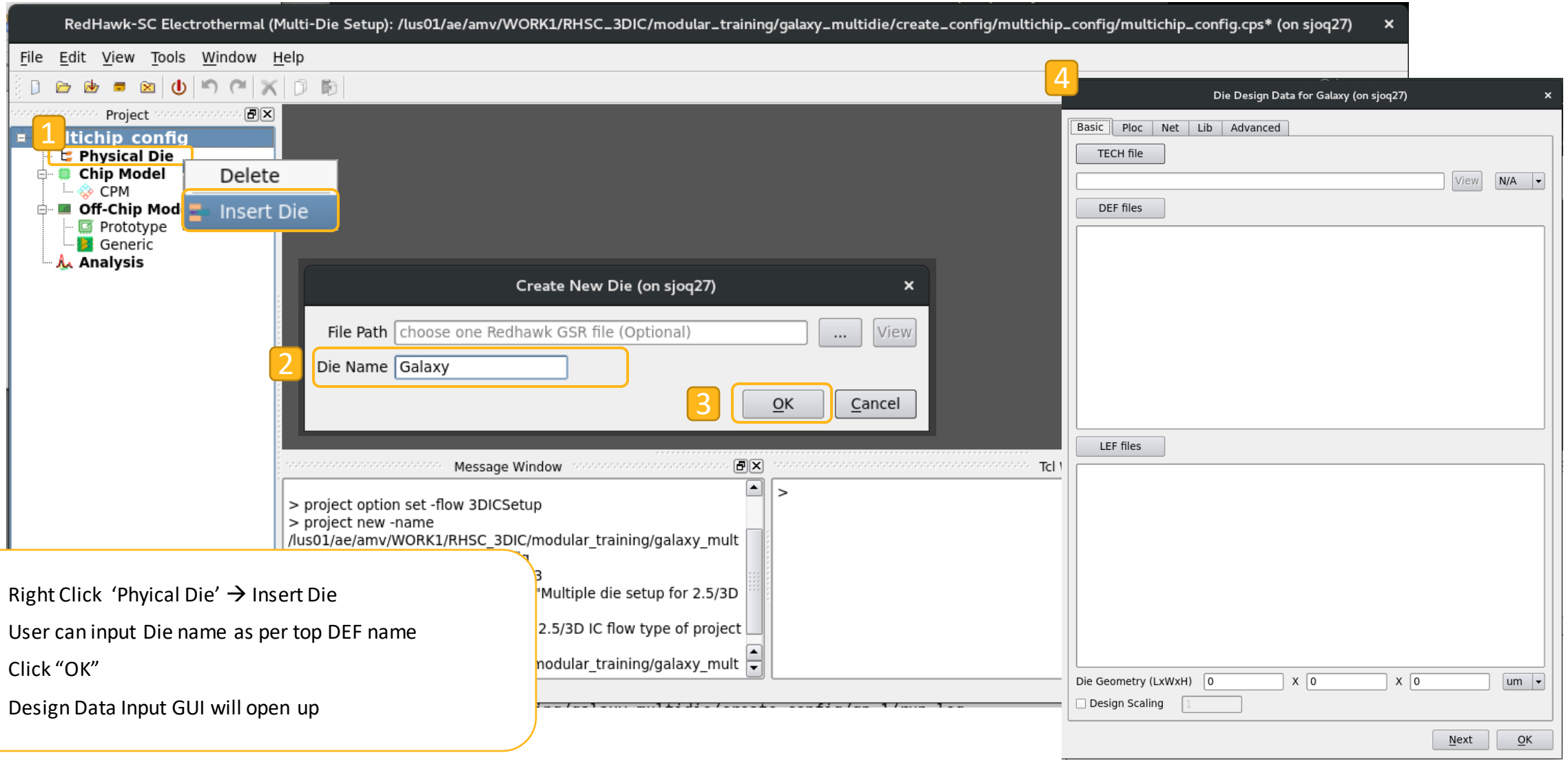
1. Select 'Tools'
2. Tools ->General
3. Set the font, project path and temporary working folder

Create a New Project



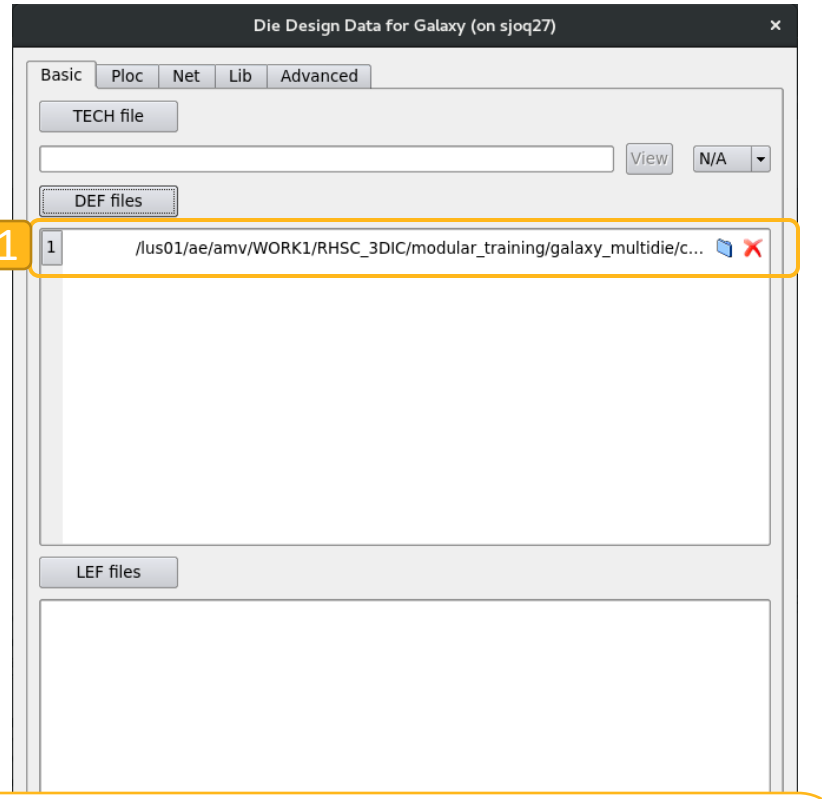
1. Select ‘File’
2. File→New
3. Input the project name and path
4. Click “OK” to create a new project

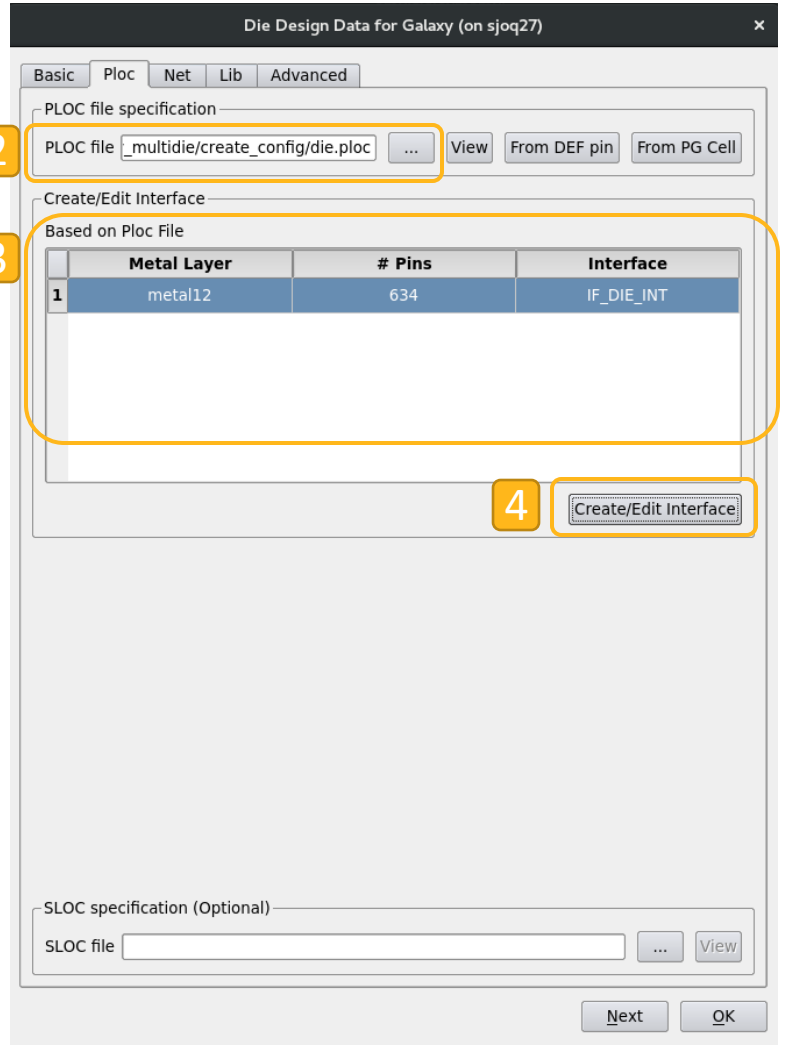
Physical Die Setting: Insert DIE



1. Right Click 'Phyical Die' → Insert Die
2. User can input Die name as per top DEF name
3. Click "OK"
4. Design Data Input GUI will open up

Physical Die Setting: Add DEF/ploc file

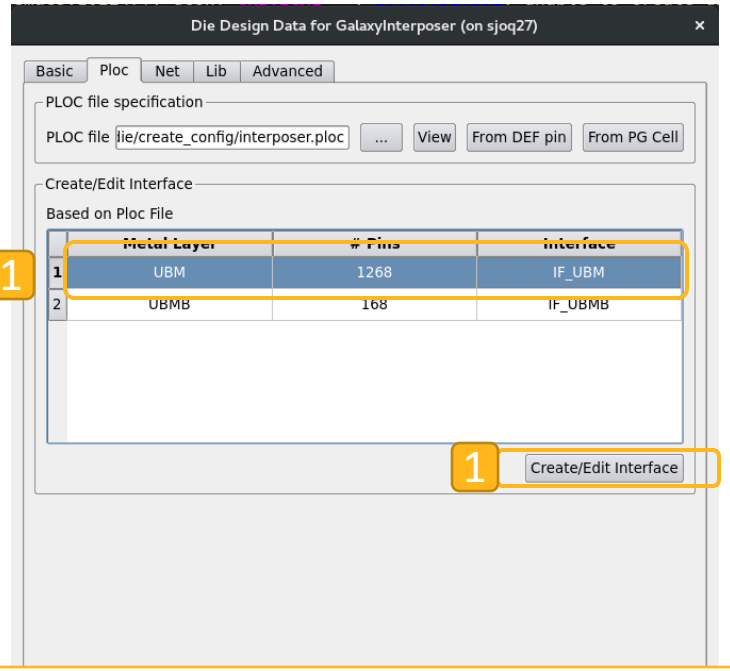


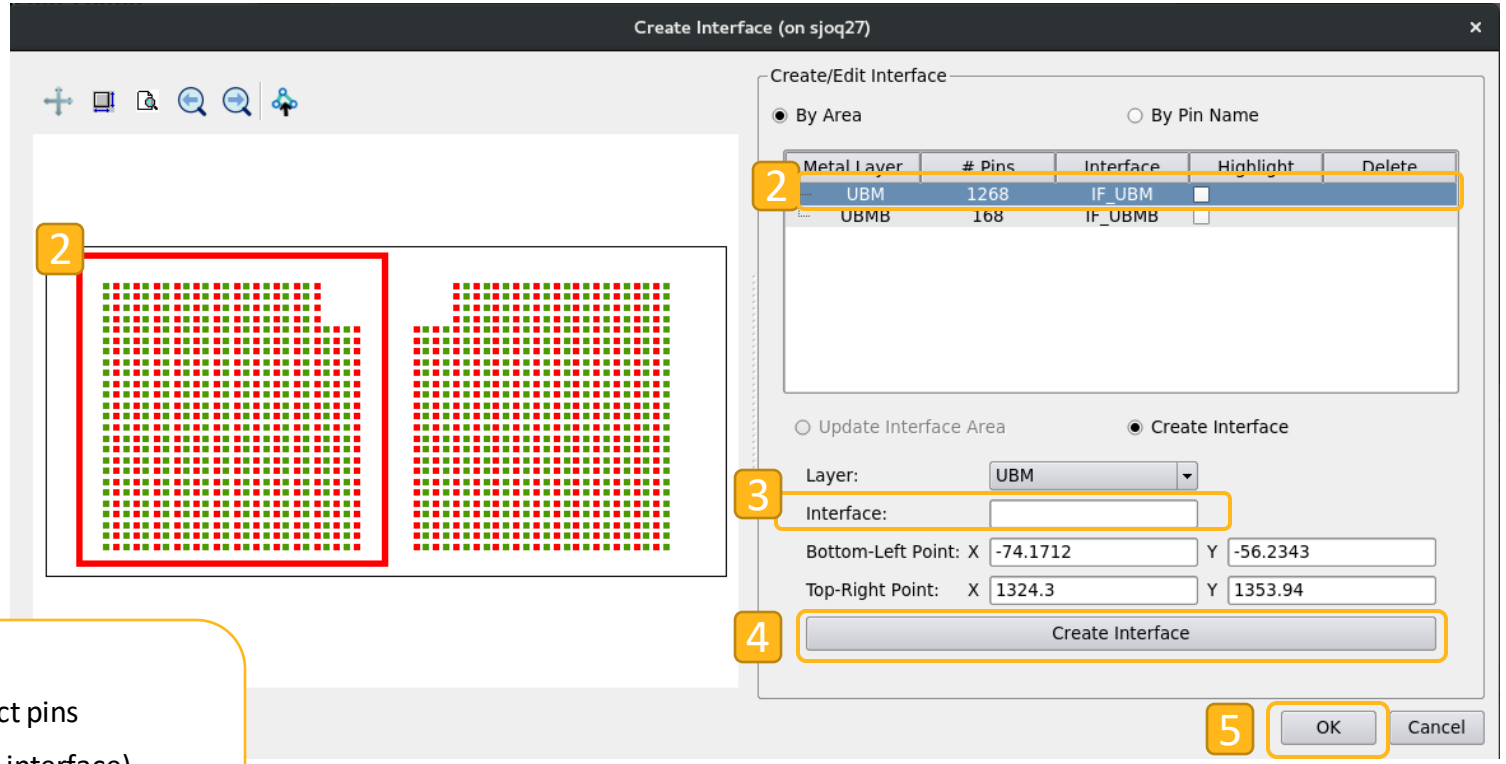


1. Add topdef file and Click Next
2. Add top file which include all interfaces pad location
3. The interfaces will be created according to different layer name.
4. If you need to modify the interface, click Create/Edit Interface

Physical Die Setting: Create Interface – by Area (1/2)

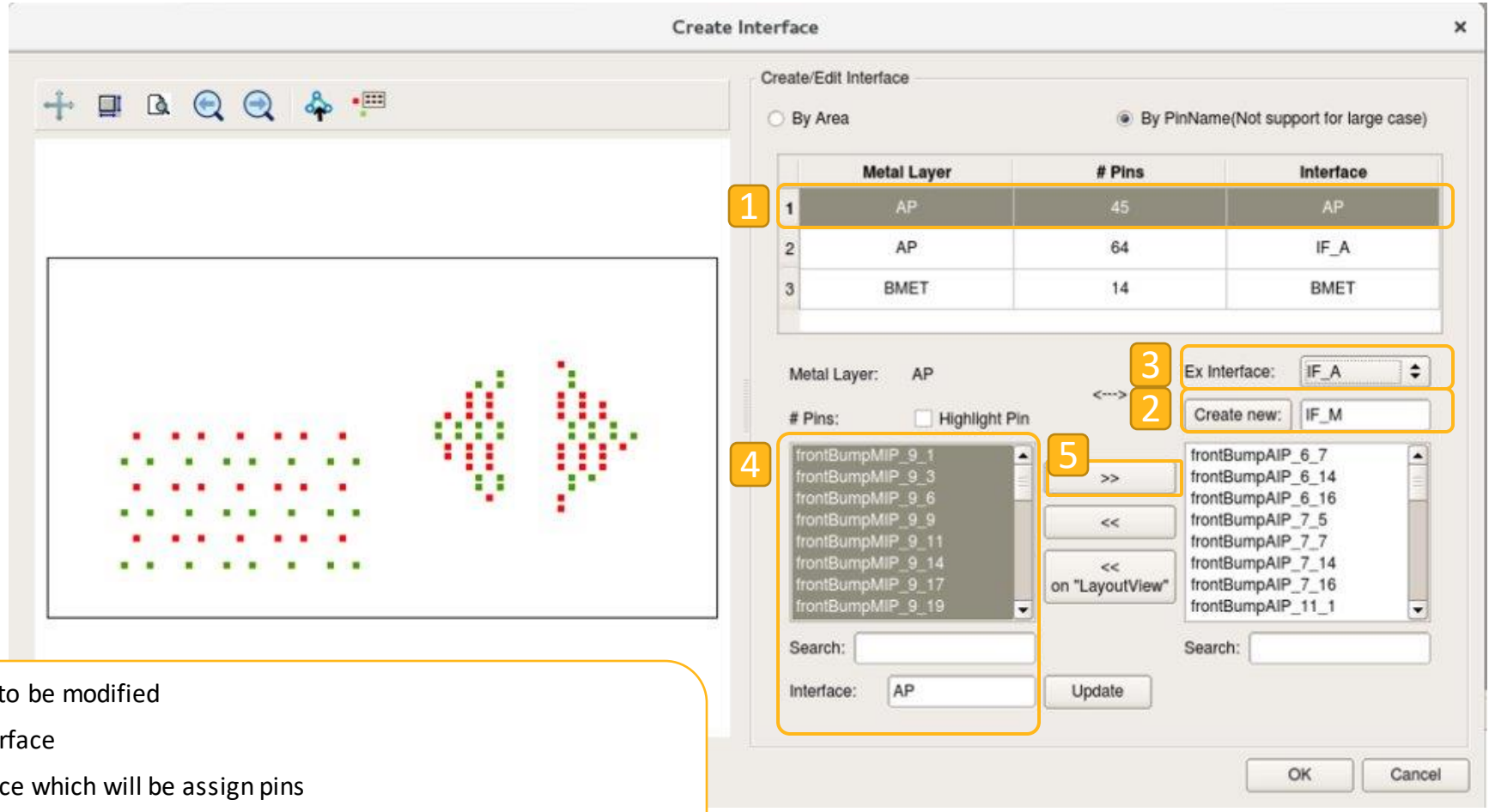
- **The die only have 1 interface, user don't need to modify**
- **Interposer has 2 interfaces to connect with DIE1 and DIE2 in layer IF_UBMB, user need to separate it.**





1. Select Interface → Create/Edit Interface
2. Select the layer & use left mouse button to pull a box to select pins
3. Input the name of new grouped interface (Interposer to DIE1 interface)
4. Create interface & repeat the process for INT_DIE2 as well as INT_C4
5. After creating separate Interface for each die connection click 'OK'

Physical Die Setting: Create Interface – by pin name(2/2)



1 Select Interface to be modified

2 Create New interface

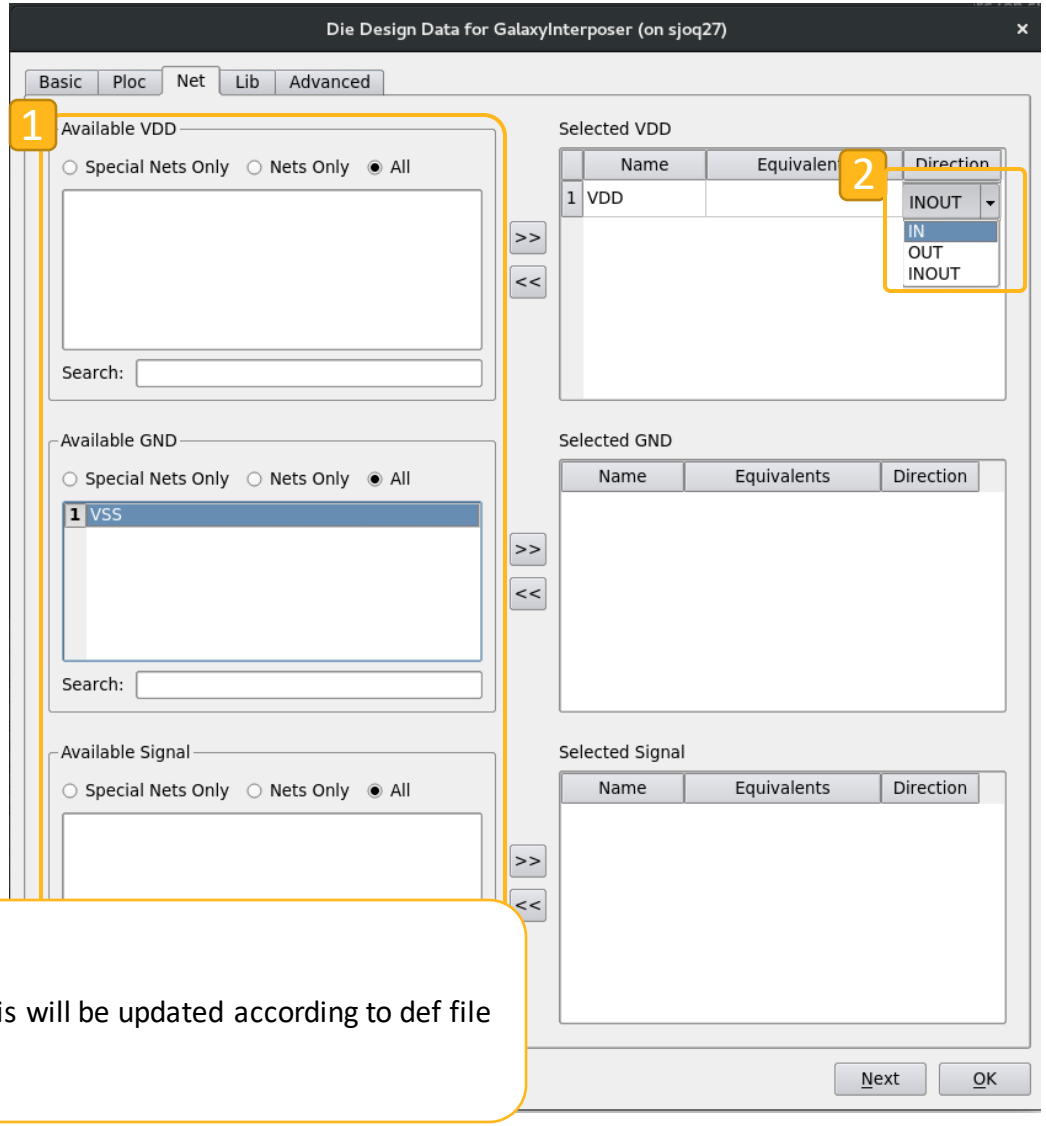
3 Select an interface which will be assign pins

4 Search pins based on the name and select pins

5 Add to the selected interface

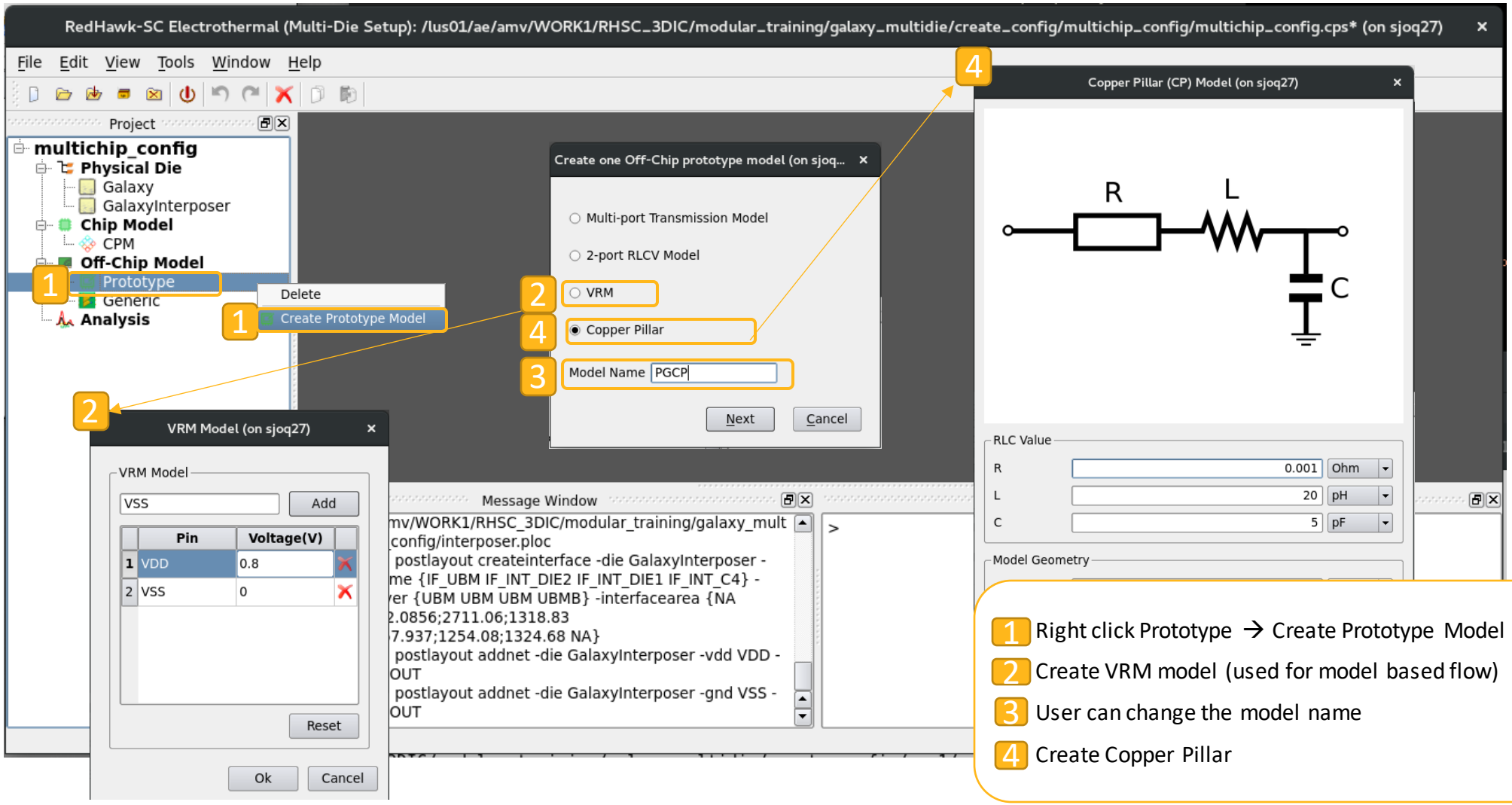
**This example is not from Galaxy 3DIC case*

Physical Die Setting: net setting



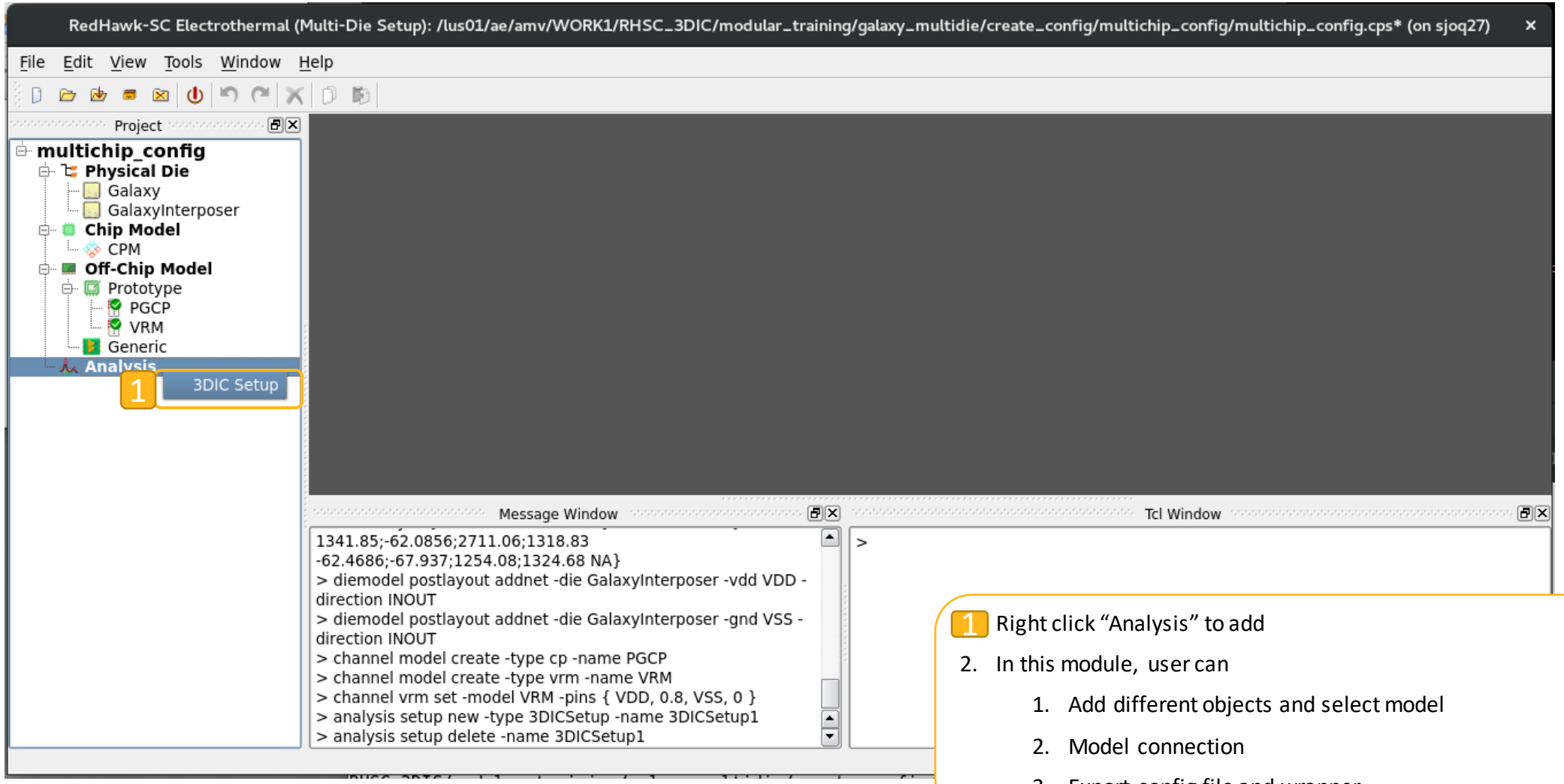
1. Select nets to be analyzed
2. Select the net direction, if def file is imported, this will be updated according to def file
3. Click 'Next' to finish physical die setting

Off-Chip Model - Prototype



1. Right click Prototype → Create Prototype Model to open GUI
2. Create VRM model (used for model based flow)
3. User can change the model name
4. Create Copper Pillar

Analysis

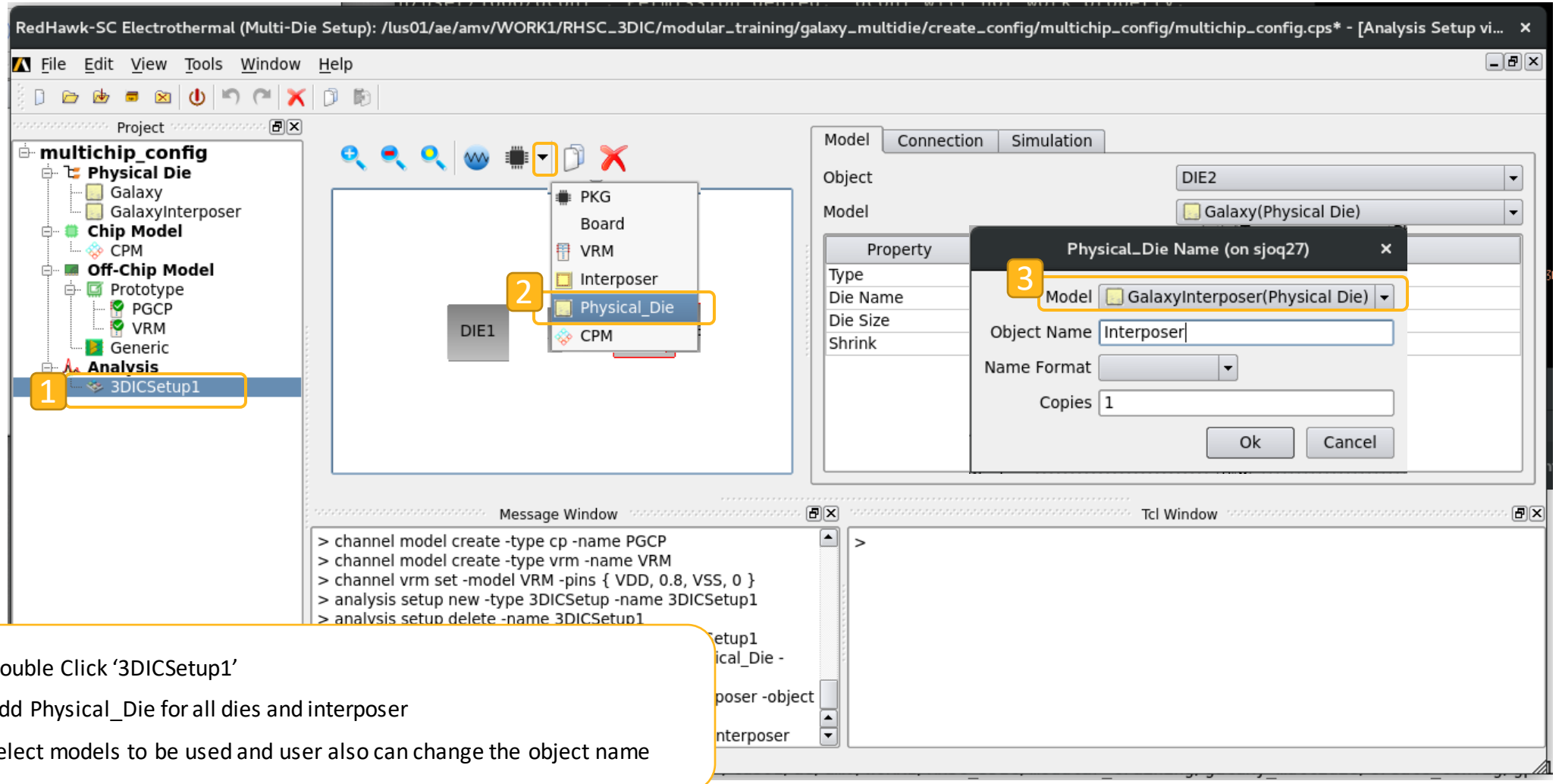


1 Right click “Analysis” to add

2. In this module, user can
    1. Add different objects and select model
    2. Model connection
    3. Export config file and wrapper

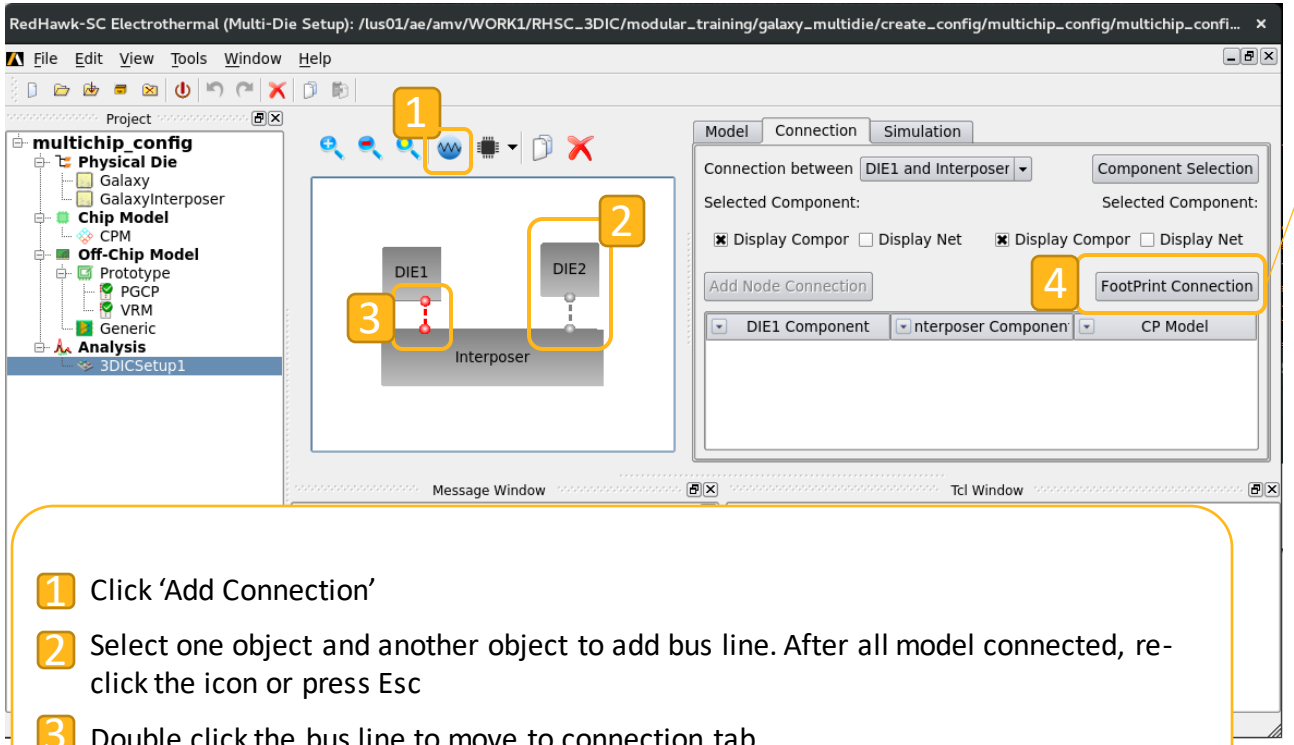
Analysis: Fulldetailed Flow (1/4)

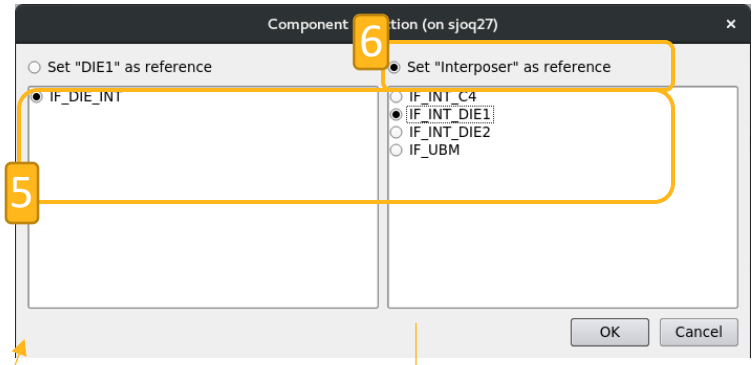
Add objects and select model

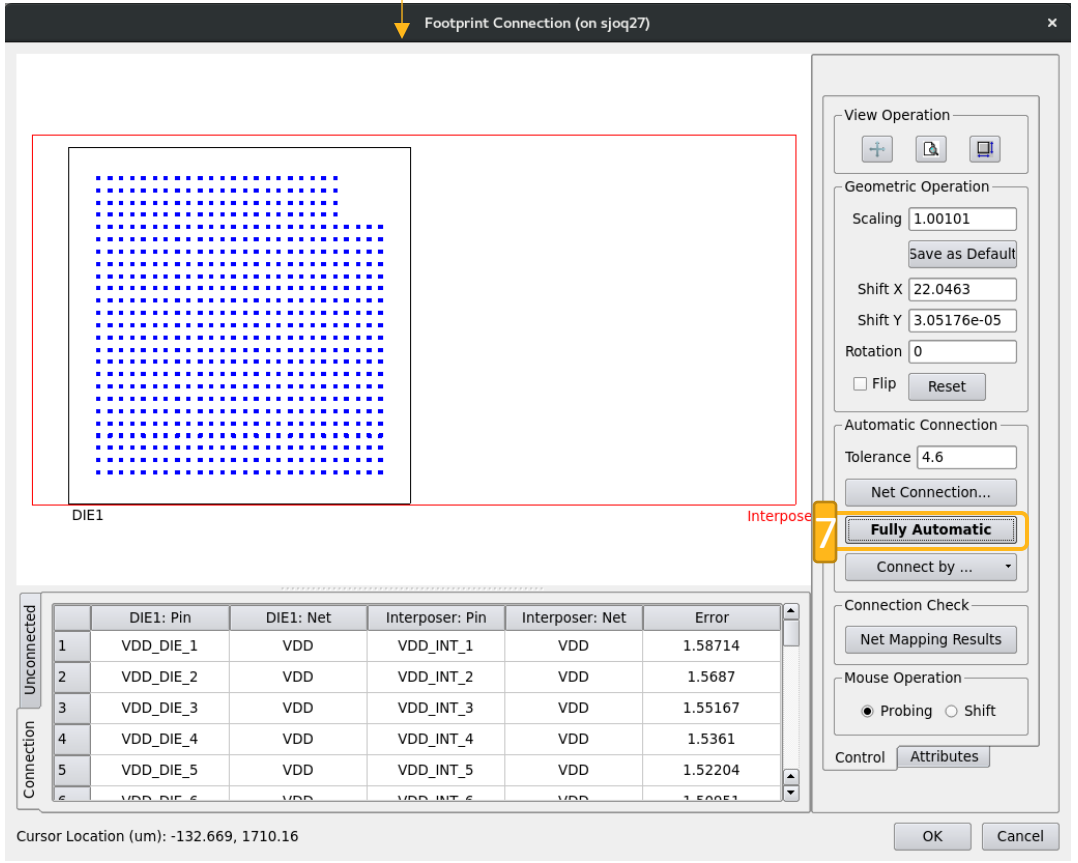


Analysis: Fulldetailed Flow (2/4)

Model Connection



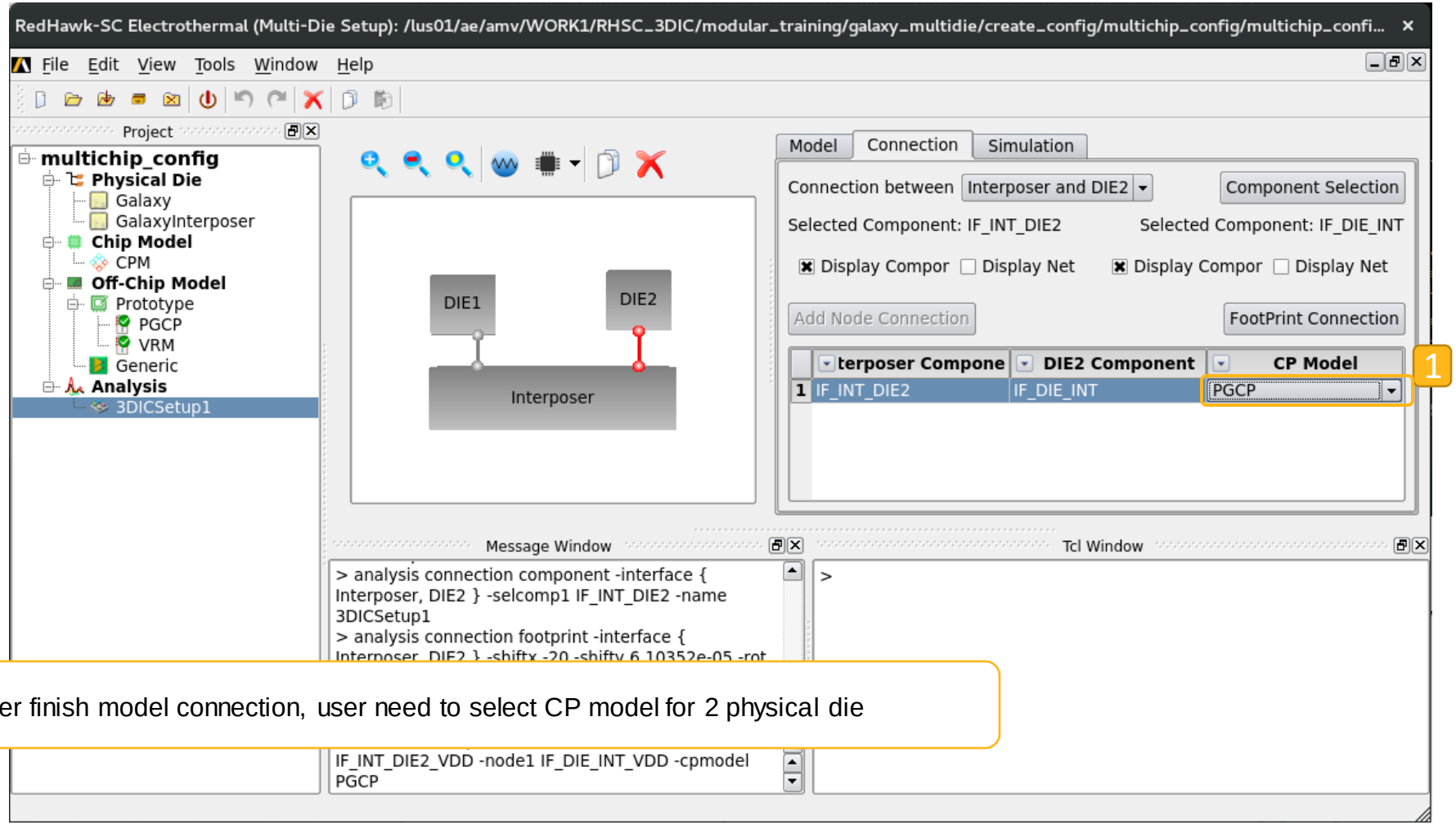




1. Click ‘Add Connection’
2. Select one object and another object to add bus line. After all model connected, re-click the icon or press Esc
3. Double click the bus line to move to connection tab
4. Click ‘Foot Print connection’
5. Select 2 interfaces to be connected
6. Make sure the object which is below another object to be taken as reference and click ‘OK’
7. Click ‘Fully Automatic’ to do die to die connection, use this method to connect all dies.

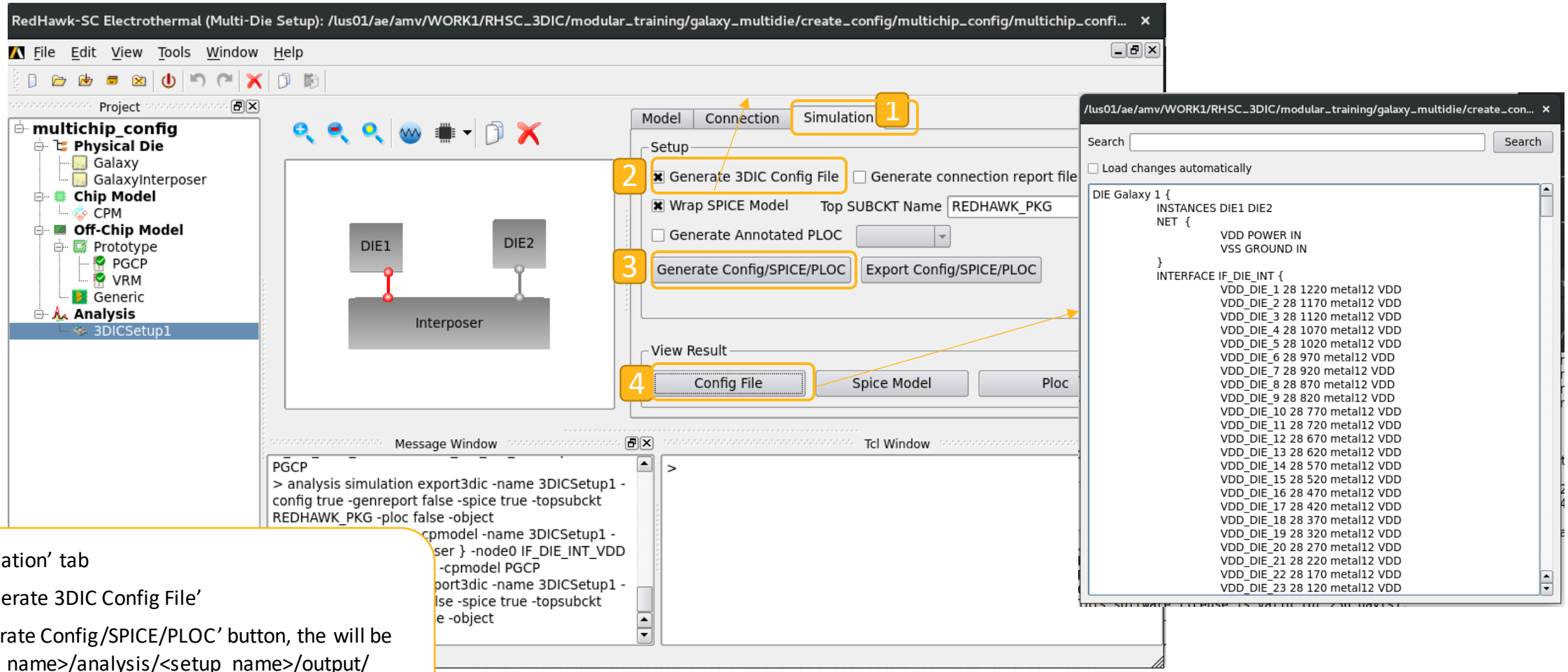
Analysis: Full Detailed Flow (3/4)

Assign CP Model



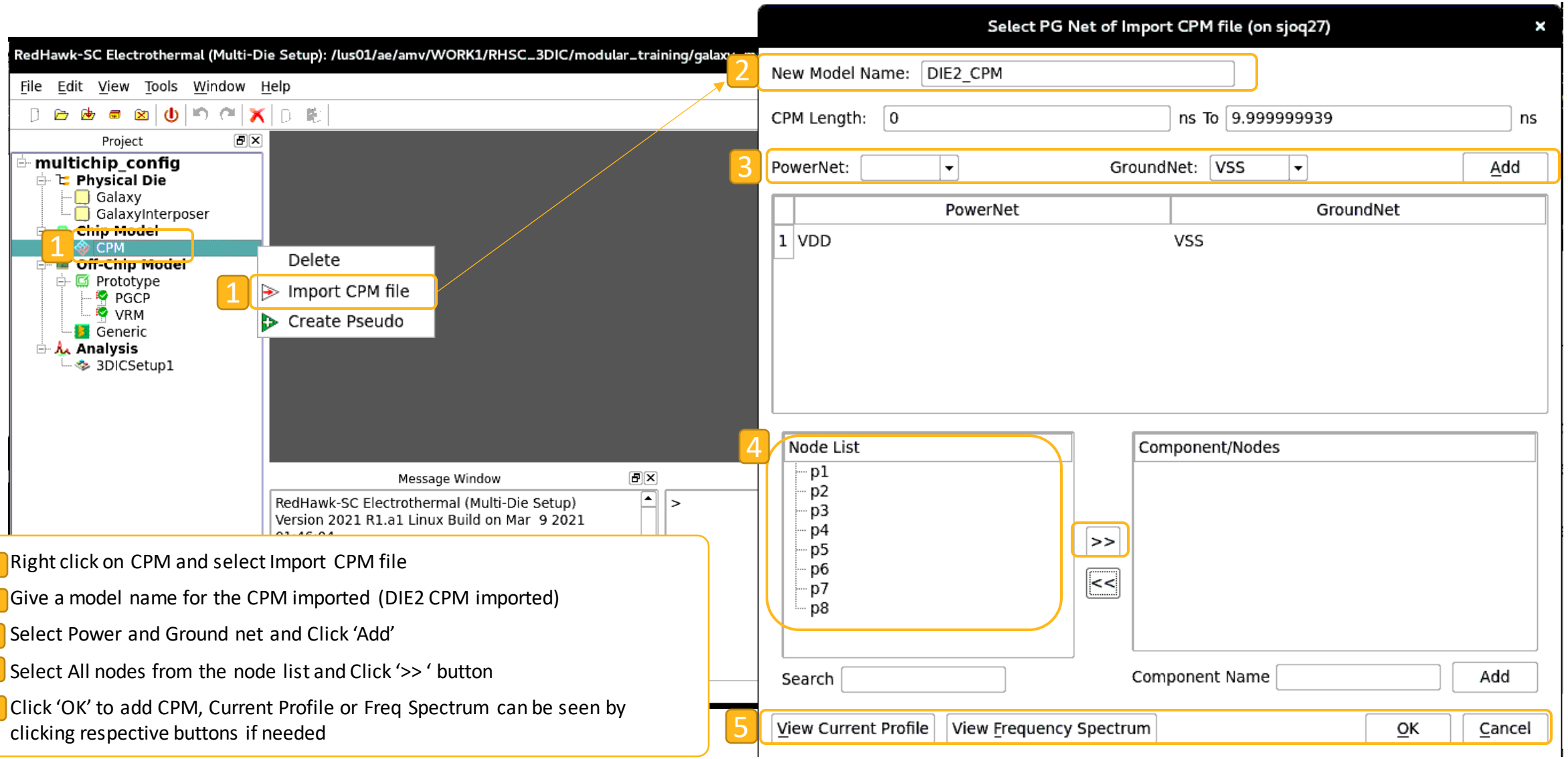
1 After finish model connection, user need to select CP model for 2 physical die

Analysis : Creating config file for full detailed analysis (4/4)



1. Click ‘Simulation’ tab
2. Select ‘Generate 3DIC Config File’
3. Click ‘Generate Config/SPICE/PLOC’ button, the will be in <project_name>/analysis/<setup_name>/output/
4. Click ‘Config file’ to view it.

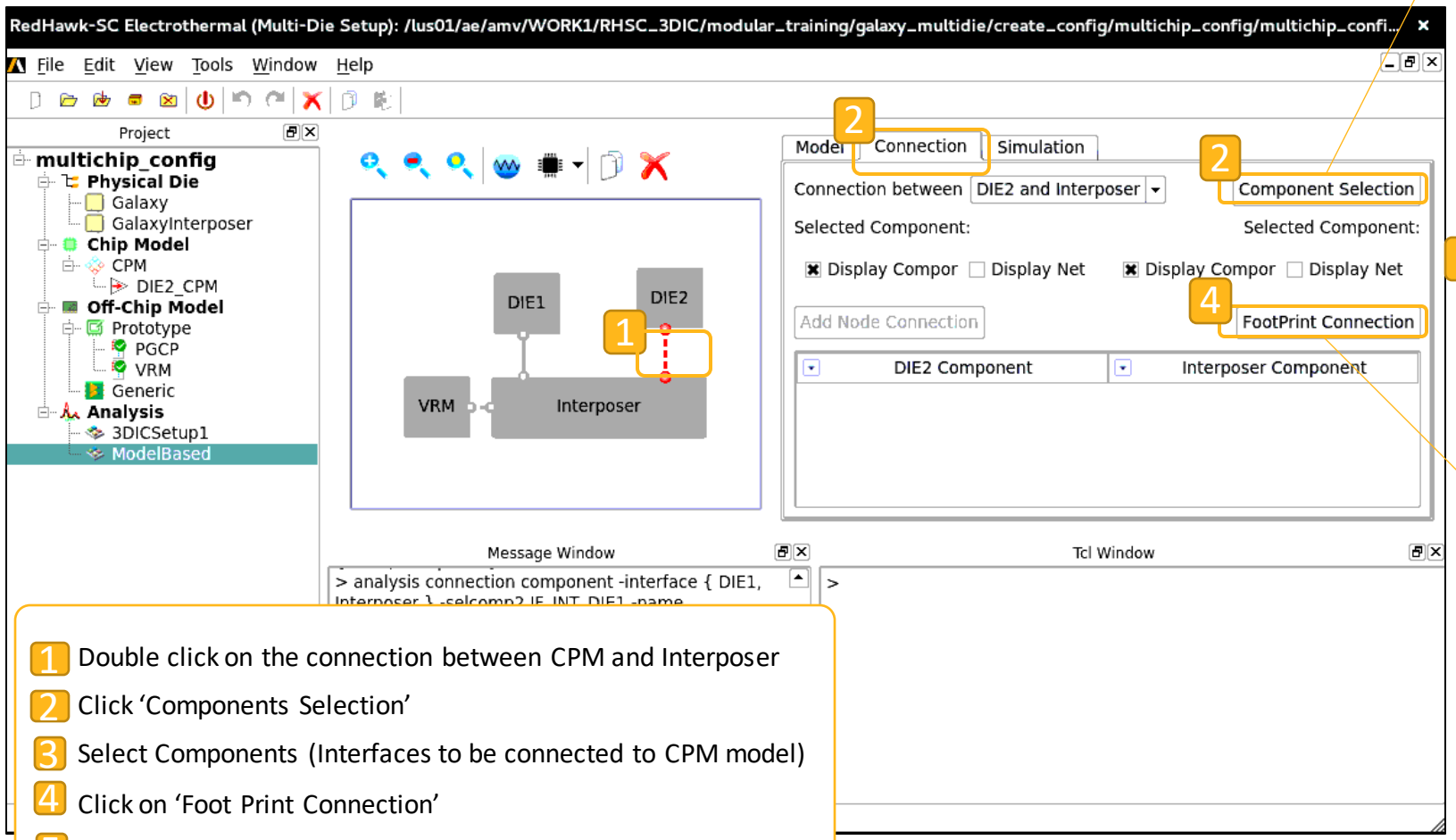
Model based flow: Importing CPM Models

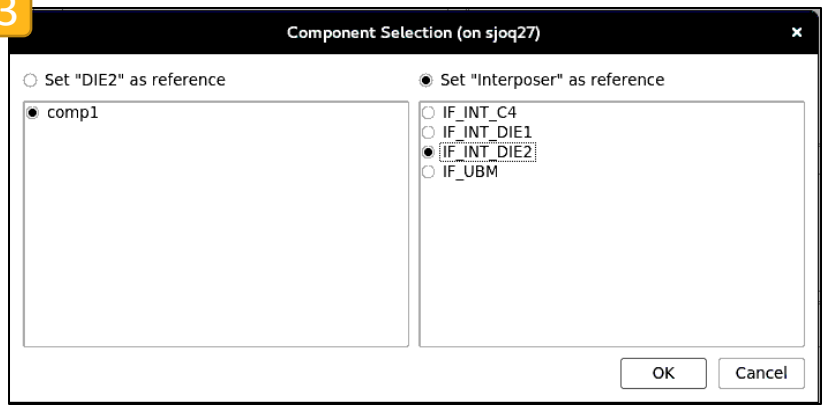


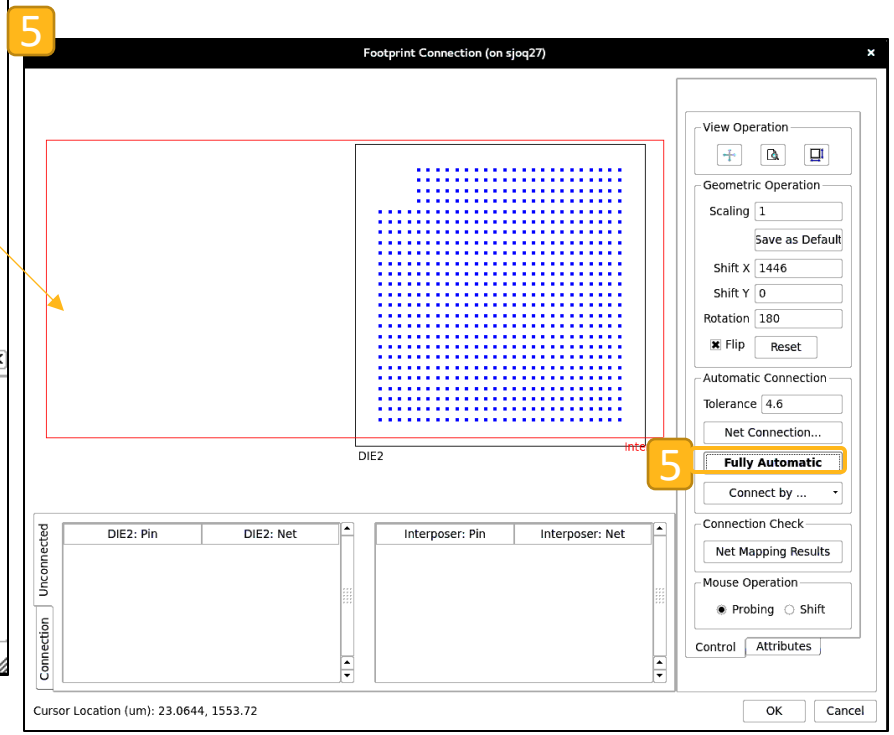
1. Right click on CPM and select Import CPM file
2. Give a model name for the CPM imported (DIE2 CPM imported)
3. Select Power and Ground net and Click 'Add'
4. Select All nodes from the node list and Click '>> ' button
5. Click 'OK' to add CPM, Current Profile or Freq Spectrum can be seen by clicking respective buttons if needed

Analysis : Model based flow (1/2)

Connecting CPM Model



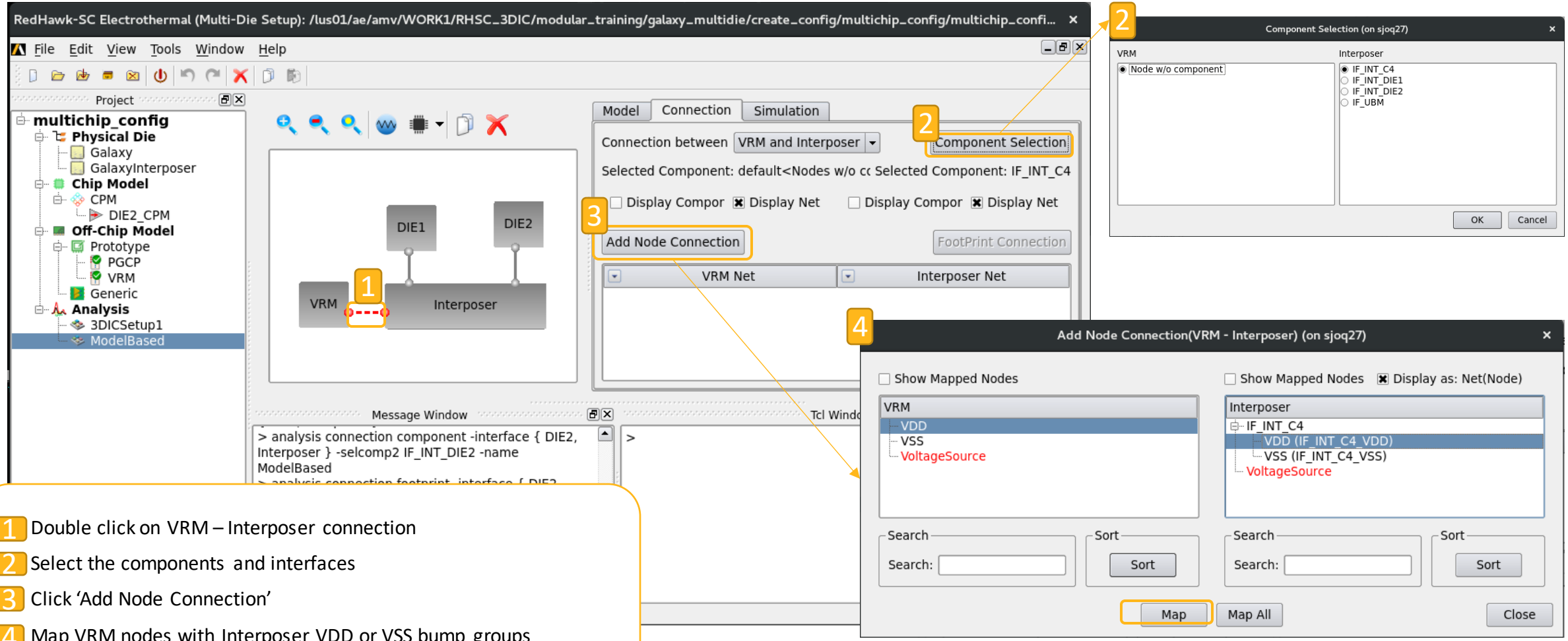




1. Double click on the connection between CPM and Interposer
2. Click 'Components Selection'
3. Select Components (Interfaces to be connected to CPM model)
4. Click on 'Foot Print Connection'
5. Fully Automatic Connection can be done here.

Analysis : Model based flow (1/2)

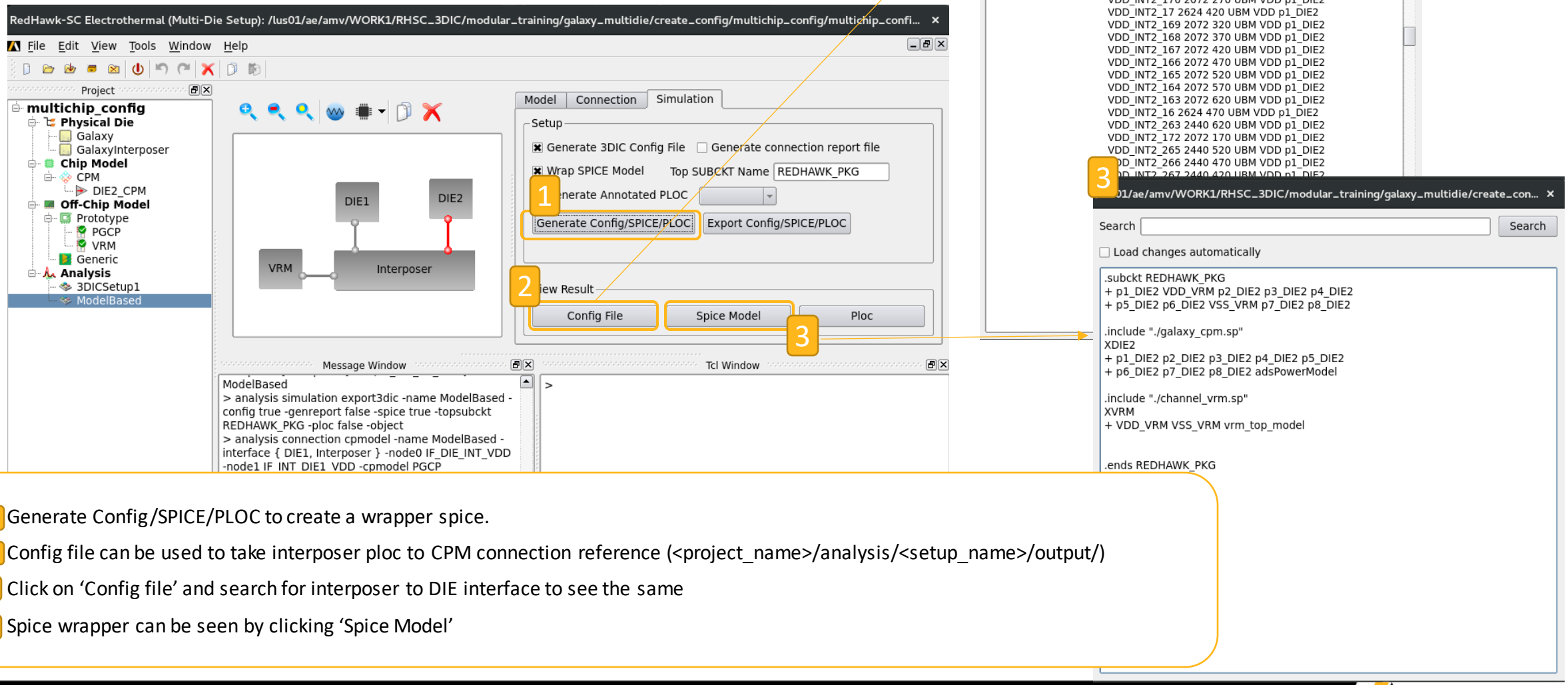
Connecting VRM Model



1. Double click on VRM – Interposer connection
2. Select the components and interfaces
3. Click 'Add Node Connection'
4. Map VRM nodes with Interposer VDD or VSS bump groups

Analysis : Model based flow (2/2)

Creating Spice Wrapper and 6 column reference ploc (config)



1. Generate Config/SPICE/PLOC to create a wrapper spice.
2. Config file can be used to take interposer ploc to CPM connection reference (<project_name>/analysis/<setup_name>/output/)
3. Click on 'Config file' and search for interposer to DIE interface to see the same
4. Spice wrapper can be seen by clicking 'Spice Model'

Multichip Analysis Usage Model

Ansys

Multichip Analysis Usage Model

Creating Config View

```
multichip_config = db.create_multichip_config_view(config_file,tag='multichip_config',options = options)
```

Creating Modified Design View

```
dv_interposer =
db.create_modified_design_view(dv0_interposer,config_view=multichip_config,die_name='GalaxyInterposer',die_sub_id=2,
options=options, tag= 'dv_interposer')
```

Creating Base Views

```
All other based views has to be created as single chip flow.
The Die specific ScenarioView, & Simulation View has to be passed to multichip analysis view
```

Creating Multi Chip Coupling View

```
coupling_views=[{'chip_name':'DIE1,'ev':ev_top},
             {'chip_name':'DIE2','ev':ev_top},
             {'chip_name':'Interposer','ev':ev_interposer},
             {'config_view':multichip_config}
]

multichip_coupling_view=db.create_multichip_coupling_view(coupling_views,tag='multichip_coupling_view')
```

Creating Multi Chip Analysis View

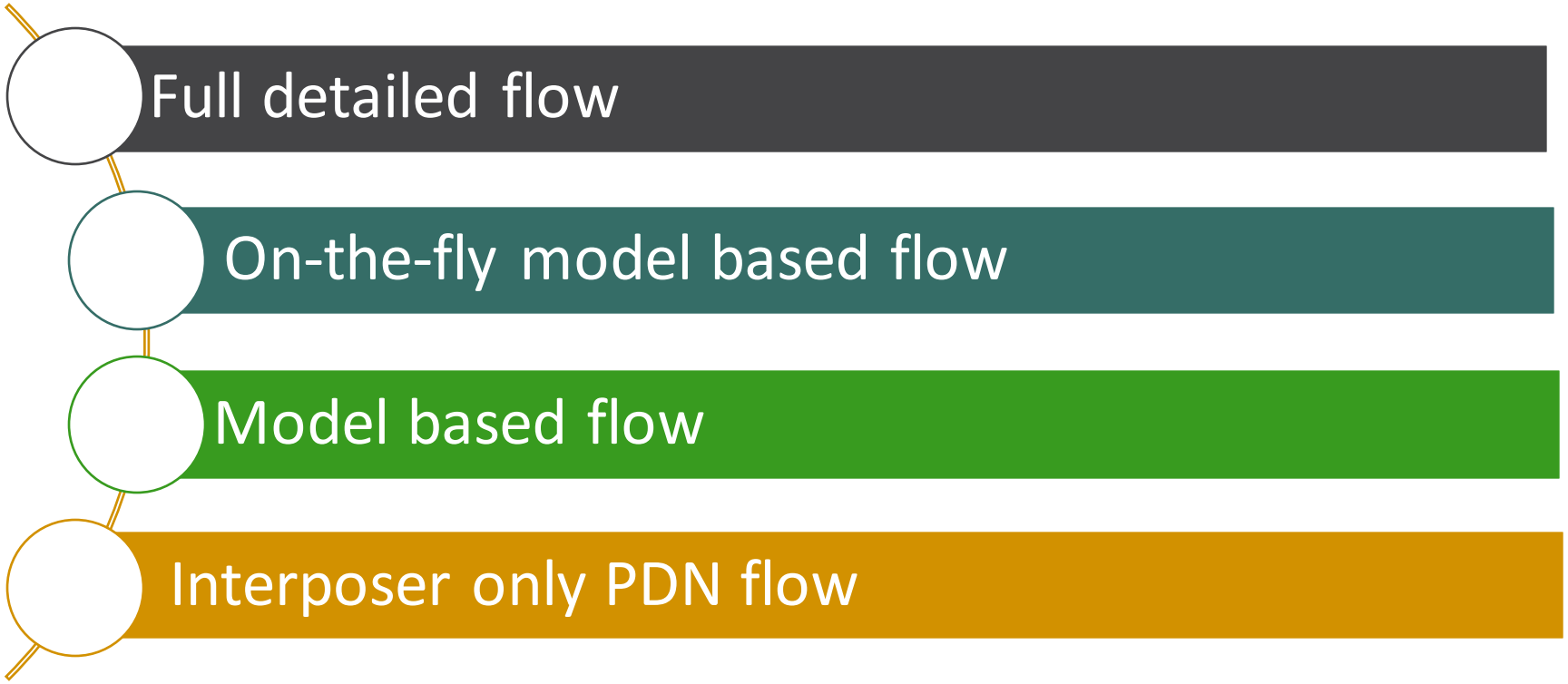
```
multichip_views = [{'chip_name':'DIE1','sv':sv_top,'scn':scn_no_prop_top, 'detailed': True},
                   {'chip_name':'DIE2','sv':sv_top,'scn':scn_no_prop_top, 'detailed': True},
                  {'chip_name':'Interposer','sv': sv_interposer, 'scn':scn_noProp_interposer, 'detailed': True},
                  {'config_view': multichip_config},
                  {'coupling_view':multichip_coupling_view},
                 ]
av_3d=db.create_multichip_analysis_view(multichip_views=multichip_views,tag='av_3d',duration=10e-9,step_size=30e-
12,options=options,keep_stats_level='Full')
```

Creating Multi Chip Power EM View

```
emv_3d_dc = db.create_multichip_electromigration_view(av_3d, tag='emv_3d_dc',options = options, mode='DC')
```

Multichip Analysis Flows

Multichip flows

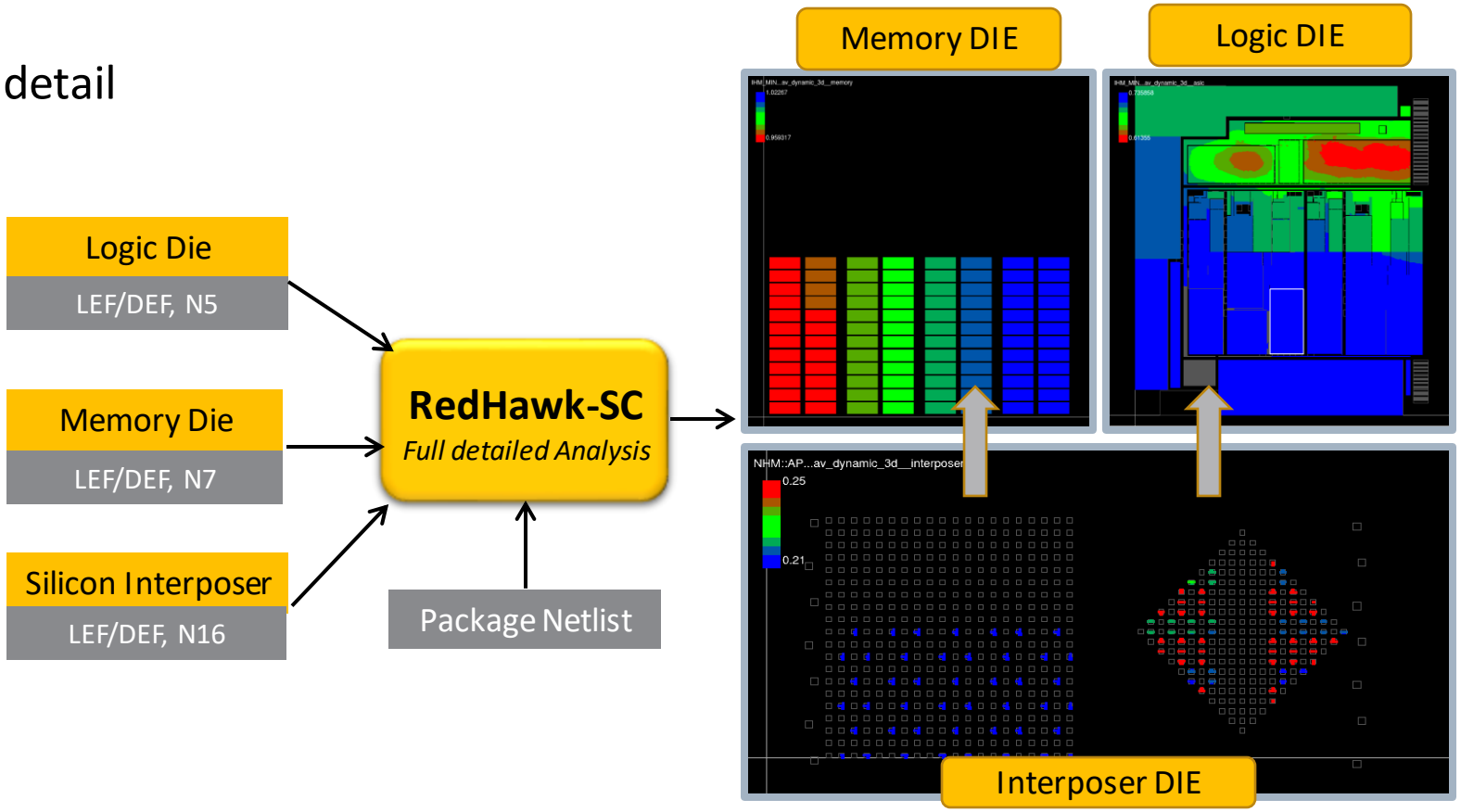


Full detailed flow

- Input multi-die design and corresponding process data (can be of different technologies), all at once
- Impact from shared P/G nets and decap in interposer die can be factored into memory and logic die
- Coupling between chips modeled in detail

Limitations

- Needs more resources (compare to other methods)
- User needs LEF/DEF for all the dies in this flow
- This may not be available, if the DIE is an IP

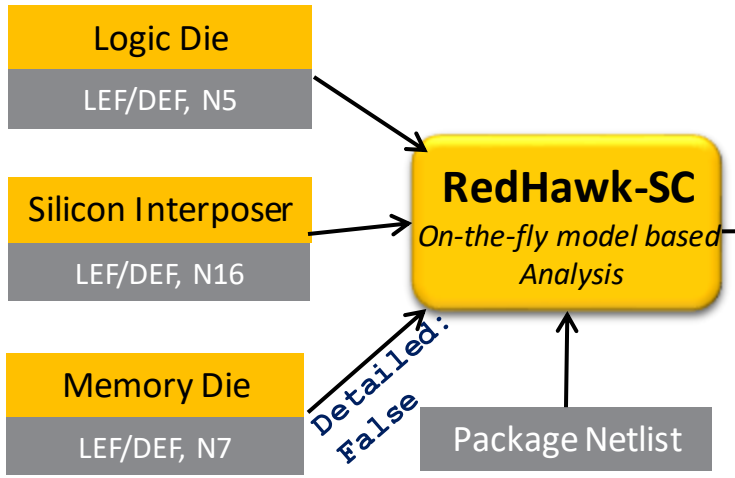


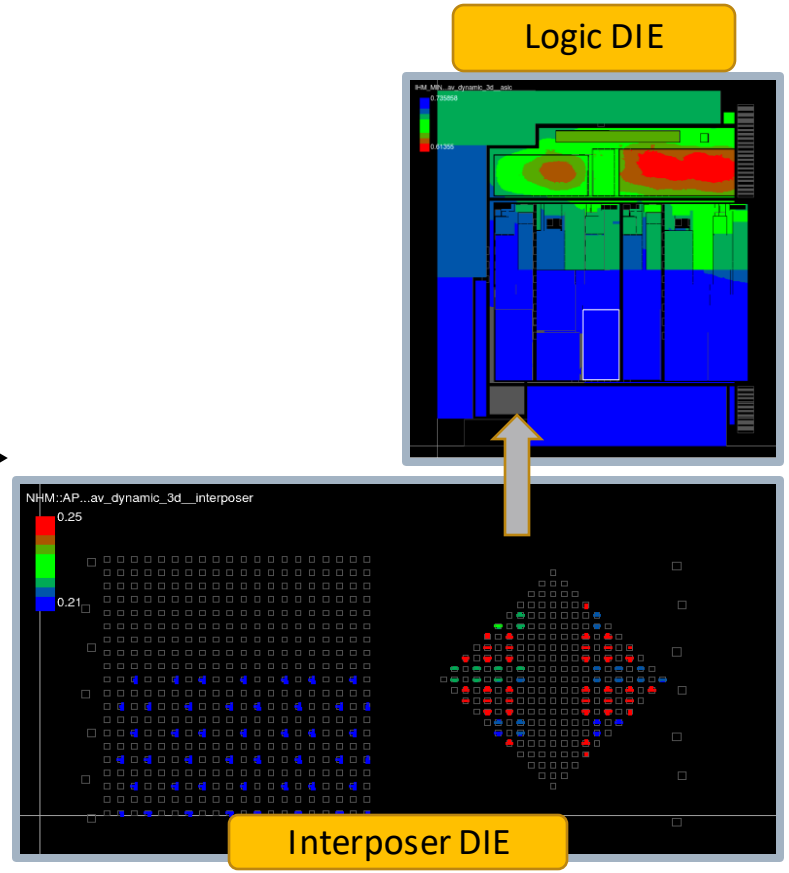
Full detailed flow - Sample flow script

```
multichip_views = [{'chip_name':'DIE1','sv':sv_top,'scn':scn_no_prop_top, 'detailed': True},#Detailed analysis for all DIE's
                   {'chip_name':'DIE2','sv':sv_top,'scn':scn_no_prop_top, 'detailed': True},
                   {'chip_name':'Interposer','sv': sv_interposer, 'scn':scn_noProp_interposer, 'detailed': True},
                   {'config_view': multichip_config},
                   {'coupling_view':multichip_coupling_view},                    ]
# MultiChip Analysis View
av_3d=db.create_multichip_analysis_view(multichip_views=multichip_views,tag='av_3d',duration=10e-9,step_size=30e-
12,options=options,keep_stats_level='Full')

av_top1=av_3d.get_chip_view(Chip('DIE1')) #get individual chip av views(single chip) using get_chip_view
av_top2=av_3d.get_chip_view(Chip('DIE2'))
av_inter=av_3d.get_chip_view(Chip('Interposer'))

# Multichip Power EM View
emv_3d_dc = db.create_multichip_electromigration_view(av_3d, tag='emv_3d_dc',options = options, mode='DC')
emv_dc_top1 = emv_3d_dc.get_chip_view(Chip('DIE1')) #get individual chip EMV views(single chip) using get_chip_view
emv_dc_top2 = emv_3d_dc.get_chip_view(Chip('DIE2'))
emv_dc_interposer = emv_3d_dc.get_chip_view(Chip('Interposer'))
```

On-the-fly model based flow

- Suitable when you need to analyze only one / few DIEs and reduce the other dies.
- Reduced die model generated by RedHawk-SC™ on fly inside the flow and used for the analysis.
- Coupling between chips modeled in detail
- Fast turn-around-time





Limitations

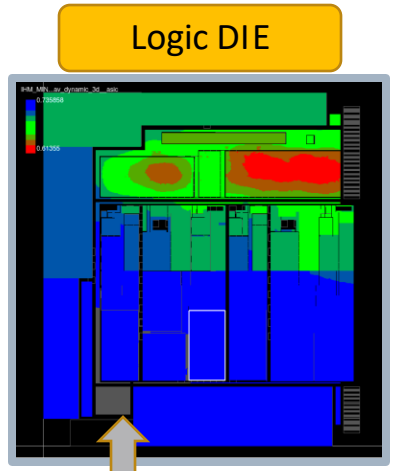
- User needs LEF/DEF for all the dies in this flow
- This may not be available, if the DIE is an IP

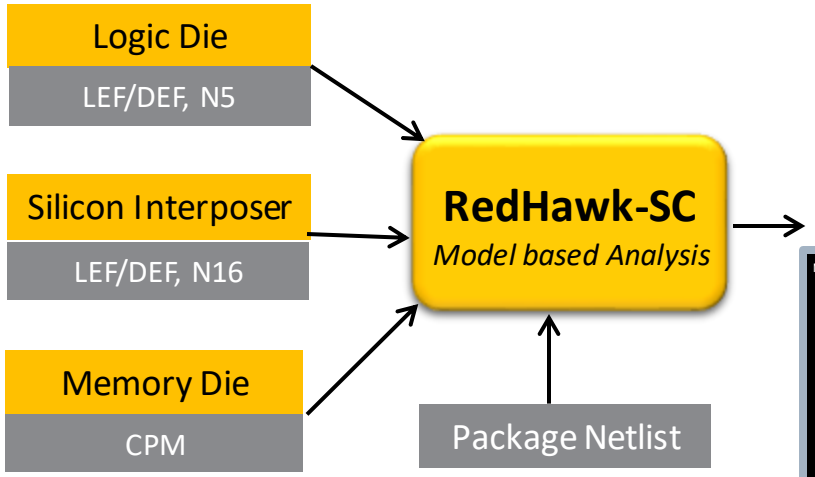
On-the-fly model based flow - Sample flow script

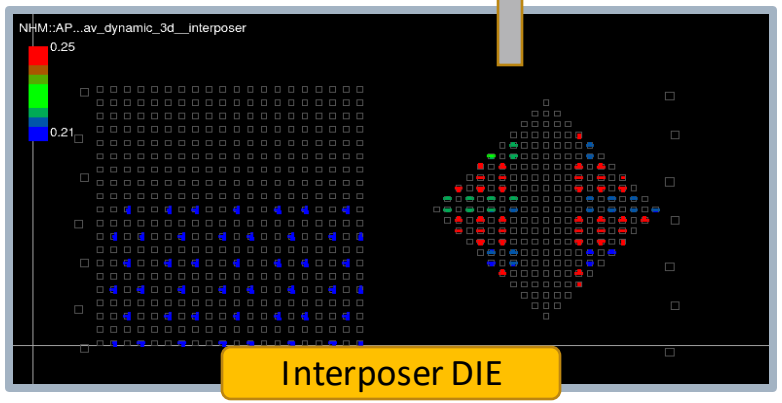
```
multichip_views = [{'chip_name':'DIE1','sv':sv_top,'scn':scn_no_prop_top, 'detailed': False},#On the fly Reduction for DIE1
                   {'chip_name':'DIE2','sv':sv_top,'scn':scn_no_prop_top, 'detailed': True},
                   {'chip_name':'Interposer','sv': sv_interposer, 'scn':scn_noProp_interposer, 'detailed': True},
                   {'config_view': multichip_config},
                   {'coupling_view':multichip_coupling_view},
                  ]

# MultiChip Analysis View
av_3d=db.create_multichip_analysis_view(multichip_views=multichip_views,tag='av_3d',duration=10e-9,step_size=30e-
12,options=options,keep_stats_level='Full')

av_top2=av_3d.get_chip_view(Chip('DIE2')) #get individual chip av views(single chip) using get_chip_view
av_inter=av_3d.get_chip_view(Chip('Interposer'))

# Multichip Power EM View
emv_3d_dc = db.create_multichip_electromigration_view(av_3d, tag='emv_3d_dc',options = options, mode='DC')

emv_dc_top2 = emv_3d_dc.get_chip_view(Chip('DIE2')) #get individual chip EMV views(single chip) using get_chip_view
emv_dc_interposer = emv_3d_dc.get_chip_view(Chip('Interposer'))
```

Model based flow

- Suitable when one/more dies are external without the complete design database
- CPM with R/L/C network and current profile, generated by RedHawk-SC™ standalone is used here.
- Enables simple hand-off and fast turn-around-time.
- Mostly useful for memory IPs where the customer will not get LEF/DEF of all the DIEs.







Limitations

- Will be difficult incase of designs without an interposer (WoW and SoIC) where all the dies are not connected to a single DIE.

Model based flow - Sample flow script

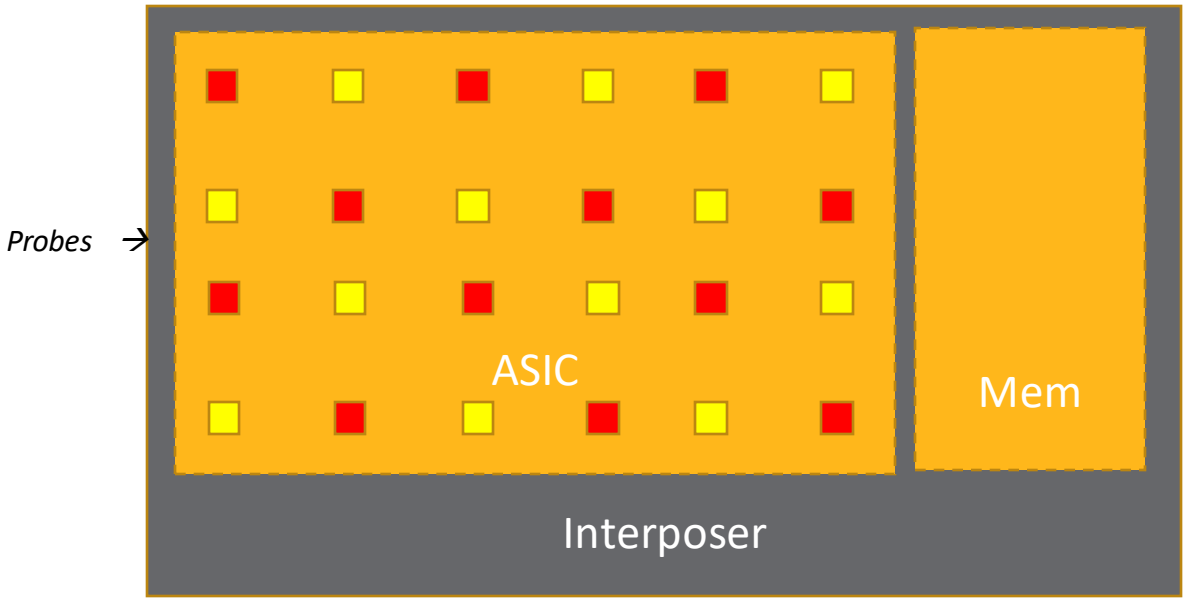
```
sv_interposer_cpm =
db.create_simulation_view(ev_interposer,enable_reduction=False,options=options,package=package.parse_ploc_spice_func('wra
pper.sp',dv=dv_interposer),tag='sv_interposer_cpm')   #Give Wrapper Package to interposer simulation view

multichip_views = [{'chip_name':'DIE1','sv':sv_top,'scn':scn_no_prop_top, 'detailed': True},
                   {'chip_name':'Interposer','sv': sv_interposer, 'scn':scn_noProp_interposer, 'detailed': True},
                   {'config_view': multichip_config},
                   {'coupling_view':multichip_coupling_view},            #CPM model used for DIE2
                  ]

# MultiChip Analysis View
av_3d=db.create_multichip_analysis_view(multichip_views=multichip_views,tag='av_3d',duration=10e-9,step_size=30e-
12,options=options,keep_stats_level='Full')

av_top1=av_3d.get_chip_view(Chip('DIE1')) #get individual chip av views(single chip) using get_chip_view
av_inter=av_3d.get_chip_view(Chip('Interposer'))

# Multichip Power EM View
emv_3d_dc = db.create_multichip_electromigration_view(av_3d, tag='emv_3d_dc',options = options, mode='DC')

emv_dc_top1 = emv_3d_dc.get_chip_view(Chip('DIE1')) #get individual chip EMV views(single chip) using get_chip_view
emv_dc_interposer = emv_3d_dc.get_chip_view(Chip('Interposer'))
```

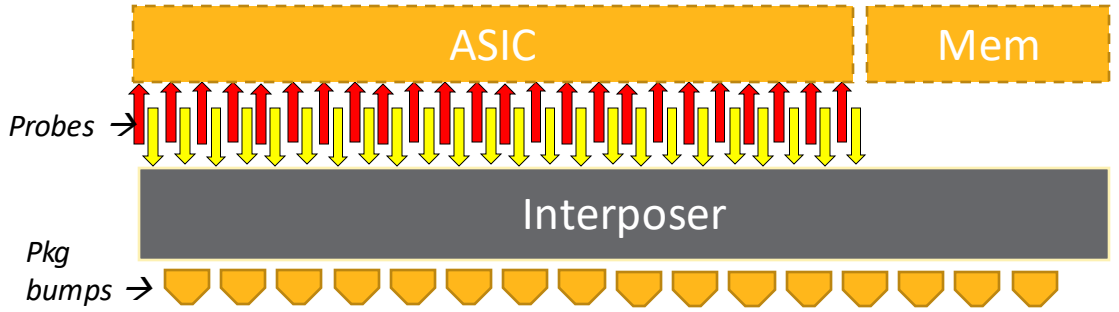
Interposer PDN quality checks (Interposer only flow)

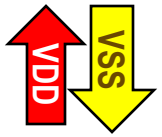
*This is a **custom flow** developed to check grid density, bump placement, connection point placements at early stage analysis.*

Only the following file is need to get the grid analysis metrics :

- ✓ ***Interposer DEF***
- ✓ ***Bump locations***
- ✓ ***Connection point locations***
- ✓ ***TSV Sub-circuit (optional)***







Static flow – with constant current

Find all the connection points which are expected to connect to the DIE and create probes on that location

Place a constant value current source at the probes.

Perform a static solve using these probes with constant currents.

Generate unique power and ground heatmaps to show the relative weaknesses

```
#Static flow with constant current source
include('create_interposer_probes.py')
multichip_config = db.create_multichip_config_view(config_file,…….)#Added to get
micro bump information

dv_interposer =
db.create_modified_design_view(dv0_interposer,eco_commands=package.parse_ploc_func('.
/interposer.ploc’ . . . . )#Plocs with C4 bumps

input_dict = {'VDD':10.0,'VSS’:-10.0} #Total bump current in static run

probes,probe_sources =
probe_creation(dv_interposer,input_dict,mcv=multichip_config,av_args=False)#Creating
probes (location , layer , net info) and probe sources(has current info)

ev_interposer =
db.create_extract_view(dv_interposer,tech_view=nv_interposer,,tag='ev_interposer,prob
es=probes)#Creating user defined probes in extraction view
tv_interposer=db.create_timing_view(dv_interposer,tag='tv_interposer’,……..)#Dummy
timing view

sv_interposer = db.create_simulation_view(ev_interposer, ……)

scn_static_int =
db.create_scenario_view(timing_view=tv_interposer,extract_view=ev_interposer,design_v
iew=dv_interposer,scenario_type="External",probe_sources=probe_sources,……)#Adding
probe sources to scenario view ,This will create external current sources at given
location(Here micro bump location)

av_static_interposer=db.create_analysis_view(sv_interposer,scn_static_int,tag='av_sta
tic_interposer’,……)
```

Dynamic flow – input current waveforms

Find all the connection points which are expected to connect to the DIE and create probes on that location

Stamp total demand or total bump current waveforms of nets at the probes.

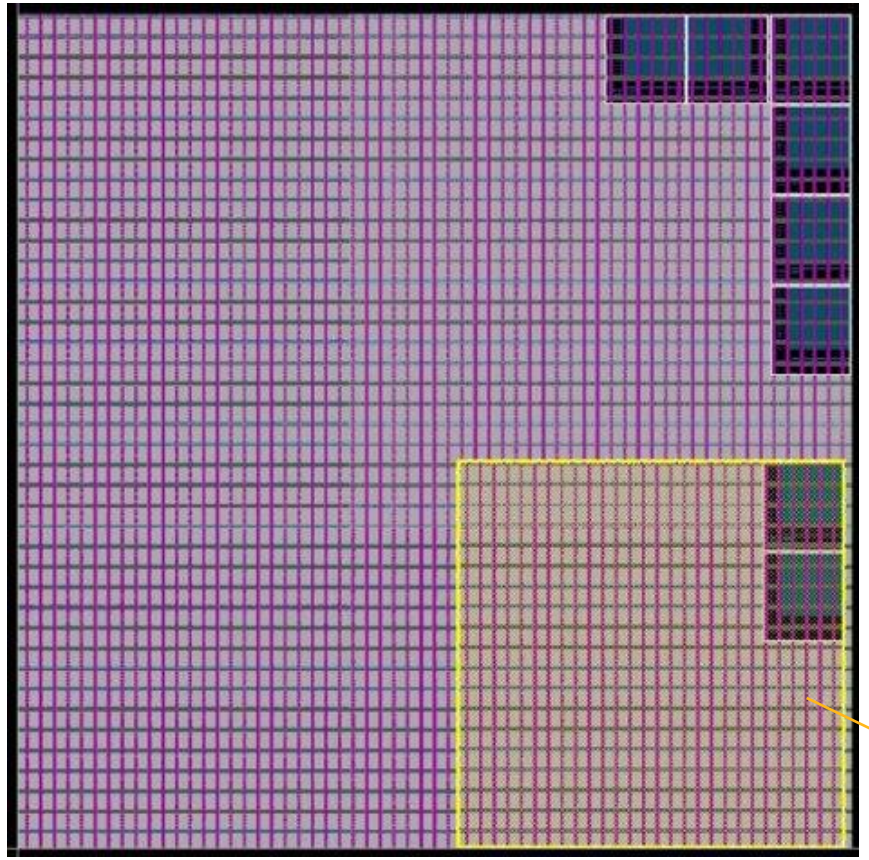
Perform Dynamic solve using these probes

Generate unique power and ground heatmaps to show the relative weaknesses

```
#Dynamic flow with current wfm's as input
include('create_interposer_probes.py')
multichip_config = db.create_multichip_config_view(config_file,……)#Added to get
micro bump information

dv_interposer =
db.create_modified_design_view(dv0_interposer,eco_commands=package.parse_ploc_func('./i
nterposer.ploc',…….)#Plocs with C4 bumps

input_dict = {Net('VDD'): Waveform([(0.0, 0.026632722467184067), (5.009999970440049e-
09, 1.645686149597168), (9.989999938397887e-09, 0.04657026007771492)]), Net('VSS'):
Waveform([(0.0, -0.026632722467184067), (5.009999970440049e-09, -1.5447595119476318) ,
(9.989999938397887e-09, -0.04929729551076889 )])}

probes,probe_sources =
probe_creation(dv_interposer,input_dict,mcv=multichip_config,av_args=False) )#Creating
probes (location , layer , net info) and probe sources(has current info)

ev_interposer =
db.create_extract_view(dv_interposer,tech_view=nv_interposer,probes=probes) )#Creating
user defined probes in extraction view
tv_interposer=db.create_timing_view(dv_interposer,……)#Dummy timing view

sv_interposer = db.create_simulation_view(ev_interposer,……)

scn_dyn_int =
db.create_scenario_view(timing_view=tv_interposer,extract_view=ev_interposer,design_vie
w=dv_interposer,scenario_type="External",probe_sources=probe_sources,……..) )#Adding
probe sources to scenario view ,This will create external current sources at given
location(Here micro bump location)

av_dyn_interposer=db.create_analysis_view(sv_interposer,scn_dyn_int,tag='av_dyn_interpo
ser',….)
```

Training labs

Training Testcase (Galaxy) Overview



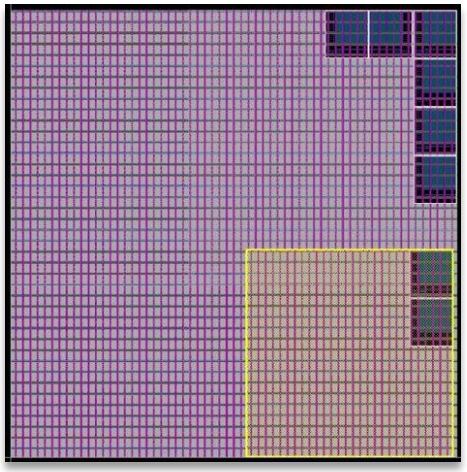
Galaxy Testcase Details	
Instance count	1.2M
Node count	5M
Domains	1 Always-ON VDD domain 1 Power-gated VDD domain 1 GND domain
#Memories	8
Tech Node	45nm
#Layers	10

Acknowledgements:

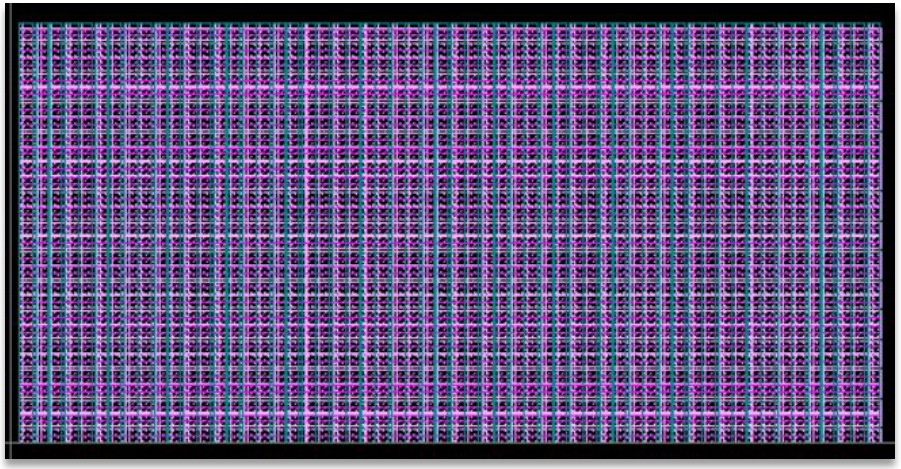
Galaxy Testcase is created using Silvaco 45nm FreePDK libraries through Si2 OpenAccess Program

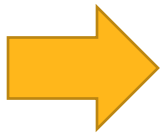
Multichip Testcase Overview

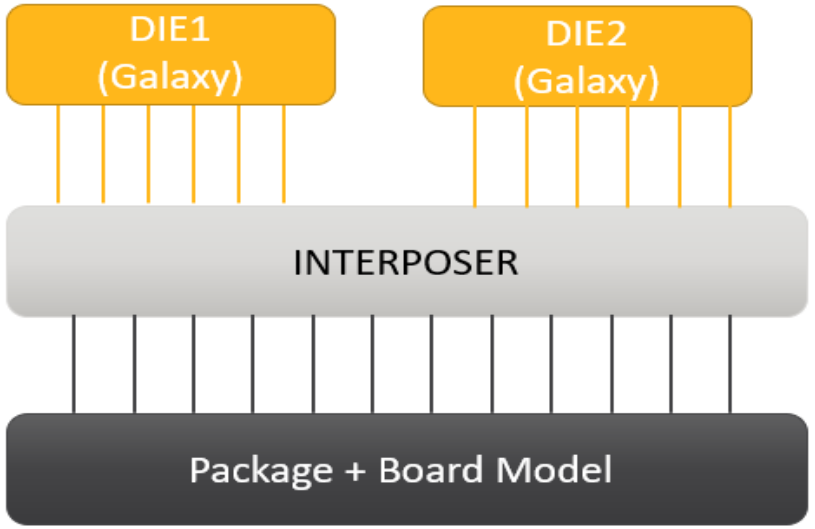
Galaxy



Interposer



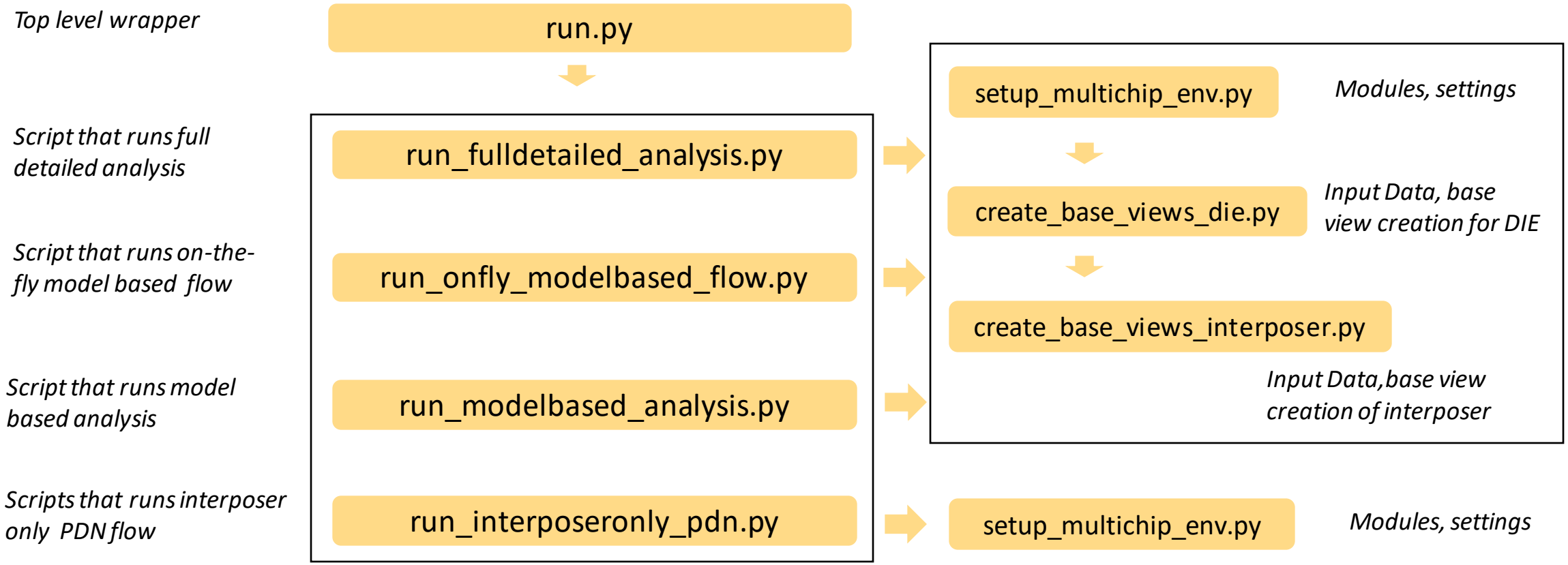




Getting Familiar with Training Lab Scripts

Lab Instructions :

- Download the Galaxy_Training.tar.gz Training Bundle
- Move to Modular_Training/14_Multichip_Analysis directory



Setting up the environment

File : scripts/setup_multichip_env.py

```
import pprint
open_scheduler_window()

ll = create_local_launcher('local')
register_default_launcher(ll, min_num_workers=10)



design_data_path = '../../design_data/'



db = gp.open_db('db')

# Auto-load view tags from an existing db
# Needed only for incremental(jump-start) runs

gp.populate_view_tags()
```

Setup for auto launching workers; here local launcher used

Set `design_data_path` variable to central design data path area

Set the DB location settings using the open_db command

Auto view-tag loading for incremental (jump-start) runs

Creating base views : DIE

File : ../scripts/create_base_views_die.py

```
include('input_files_die.py')

lv_top = db.create_liberty_view(liberty_file_names=lib_files_top,apl_switch_files=
switch_files_top,options=options,tag = "lv_top")

nv_top = db.create_tech_view(tech_file_name=tech_file_top,options=options,tag = "nv_top")

dv_top =
db.create_modified_design_view(design_view=dv0_top,config_view=multichip_config,die_name=
'Galaxy',die_sub_id=1,options=options,tag = "dv_top")

ev_top = db.create_extract_view(design_view = dv_top,tech_view = nv_top,calculate_spr =
True,temperature=25,options=options,tag = "ev_top")

.. .. ..
```

DIE : Input data base

Base Views Creation For Individual DIEs

Creating base views : Interposer

File : ../scripts/create_base_views_interposer.py

```
include('input_files_interposer.py')

nv_interposer =
db.create_tech_view(tech_file_name=tech_file_interposer,options=options,tag='nv_int
erposer',via_variation='typical')

dv_interposer =
db.create_modified_design_view(dv0_interposer,config_view=multichip_config,die_name
='GalaxyInterposer',die_sub_id=2,options=options, tag= 'dv_interposer')

ev_interposer =
db.create_extract_view(dv_interposer,tech_view=nv_interposer,temperature=25,options
=options,tag='ev_interposer',calculate_spr=True,extract_inductance=True)

.. .. ..
```

INTERPOSER : Input data base

Base Views Creation for interposer

Overview of lab

Files : scripts/input_files_die.py
scripts/input_files_interposer.py

```
def_files = [

design_data_path + '/defs/Galaxy.def'

.. .. ..

]


lef_files = [

design_data_path +
'/lefs/switch_cell.lef',

.. .. ..

]
```

Specify all the design files in input_files.py

Files : scripts/create_base_views_die.py
scripts/create_base_views_interposer.py

```
include('input_files_die.py')

dv_top =
db.create_modified_design_view(de
sign_view=dv0_top,config_view=mul
tichip_config,die_name='Galaxy',d
ie_sub_id=1,options=options,tag =
"dv_top")

ev_top =
db.create_extract_view(design_vie
w = dv_top,tech_view =
nv_top,calculate_spr =
True,temperature=25,options=optio
ns,tag = "ev_top")

.. .. ..
```

Create base views of DIE and interposer

Files : scripts/run_fulldetailed_analysis.py
scripts/run_onfly_modelbased_flow.py
scripts/run_modelbased_analysis.py

```
include('setup_multichip_env.py')
multichip_config =
db.create_multichip_config_view(config
_file,tag='multichip_config',options =
options)

include('create_base_views_die.py')
include('create_base_views_interposer.
py')

multichip_coupling_view=db.create_mult
ichip_coupling_view(coupling_views,tag
='multichip_coupling_view',
options=options)

av_3d=db.create_multichip_analysis_vie
w(multichip_views=multichip_views,tag=
'av_3d',duration=10e-9,step_size=30e-
12,options=options,keep_stats_level='F
ull')
```

Running the multichip flows

Launching RedHawk-SC - Get started with your labs

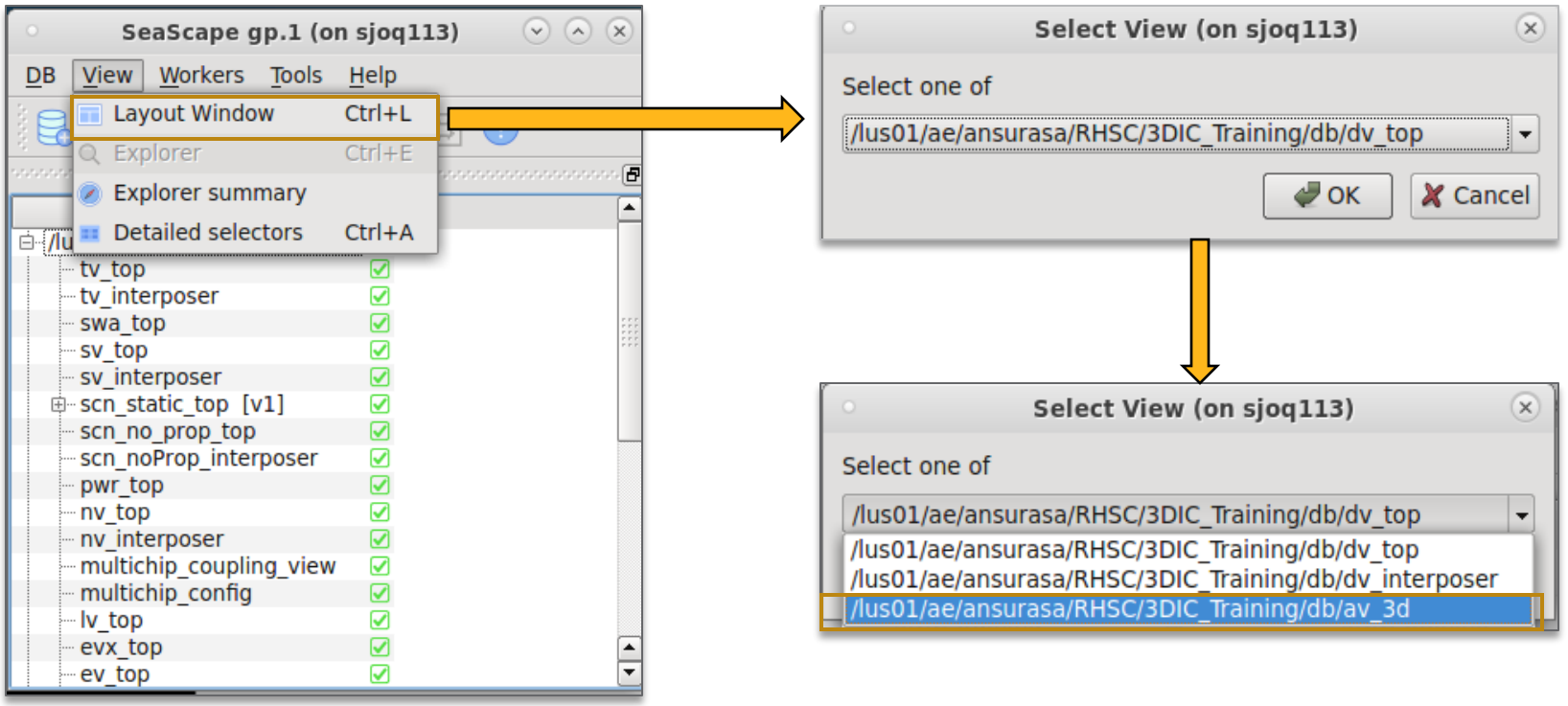
- Batch mode execution example:
  - `<path_to_rhsc_installation>/bin/redhawk_sc run.py`
- Interactive mode execution example:
  - `<path_to_rhsc_installation>/bin/redhawk_sc -i`
  - It needs an exit() command in script or entered manually to exit the Python shell
- Connecting to a live RedHawk-SC run:
  - RedHawk-SC allows querying of data/results from an active session, by remotely attaching to the session
  - Multiple users can attach to the same session from multiple machines for querying/viewing results
  - `<path_to_rhsc_installation>/bin/redhawk_sc -r <gp_dir>`
- Execution Log Files:
  - All RHSC log files reside by default in gp<> folder
    - If the run is fired in the same directory, tool will create gp.1, gp2 incrementally
    - 'latest.gp' link will point to the most recent gp directory
  - Main log file for RedHawk-SC would be <gp_directory>/run.log file.

RHSC Multichip Analysis
GUI Features

Ansys

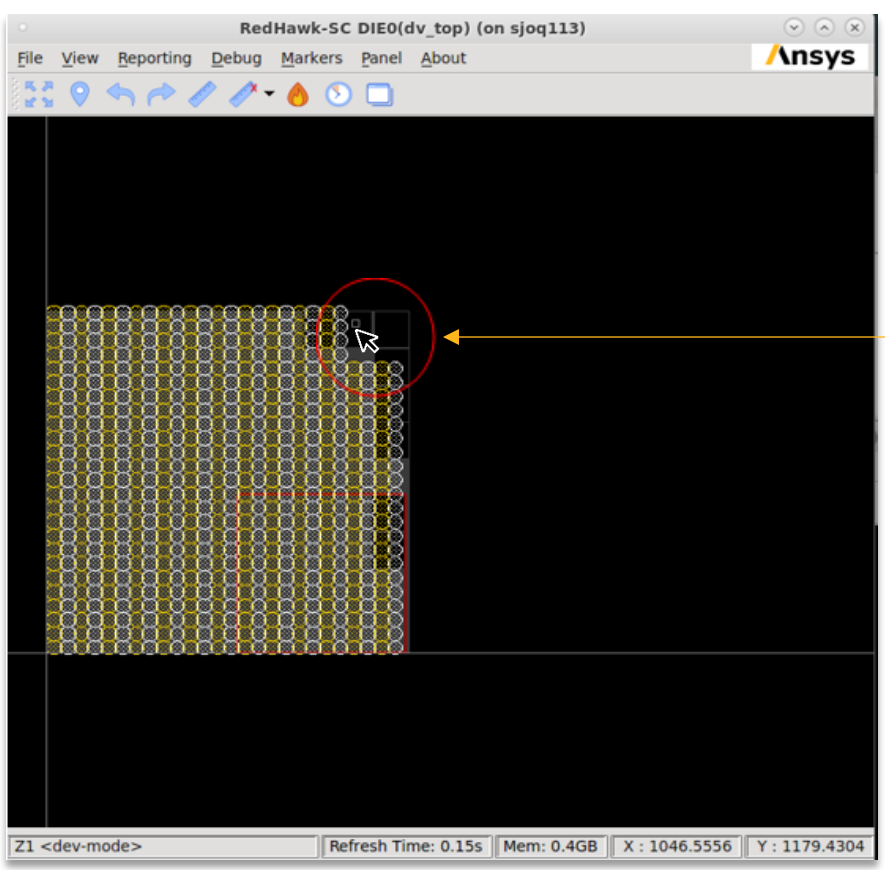
Opening GUI

View → Layout Window → from drop down select **multichip analysis view**



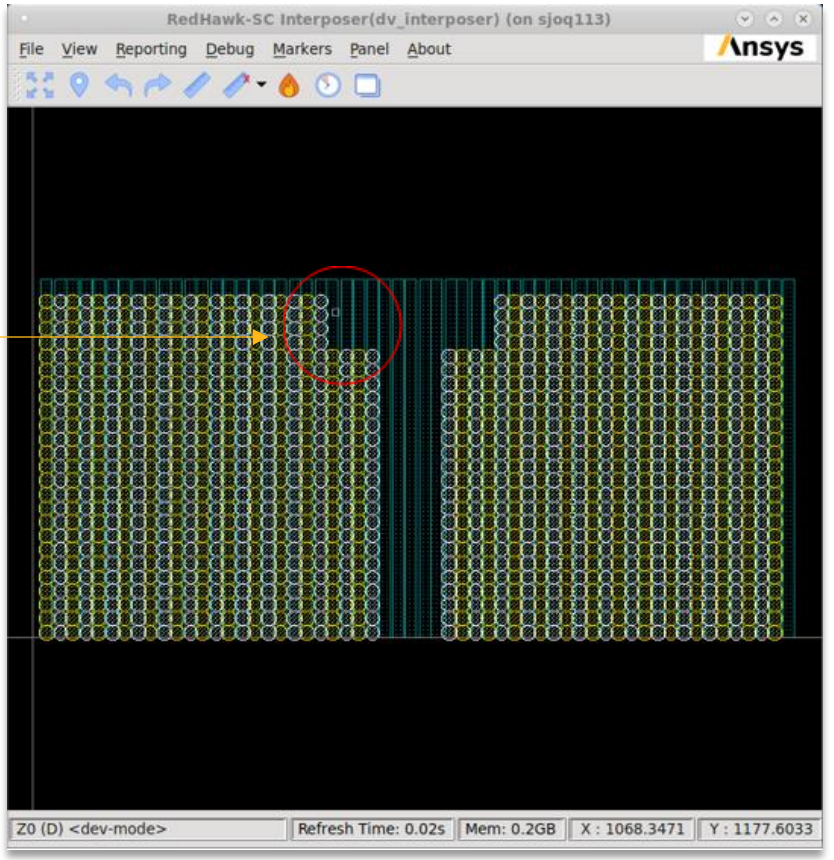
Cross Wire Tracking

Mouse pointer being tracked live over the multiple GUIs, and which will help to understand the relative locations (die orientation) of each DIE.



DIE1 GUI

Relative location of mouse pointer being tracked as a box



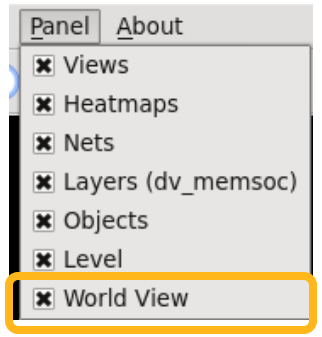
INTERPOSER GUI

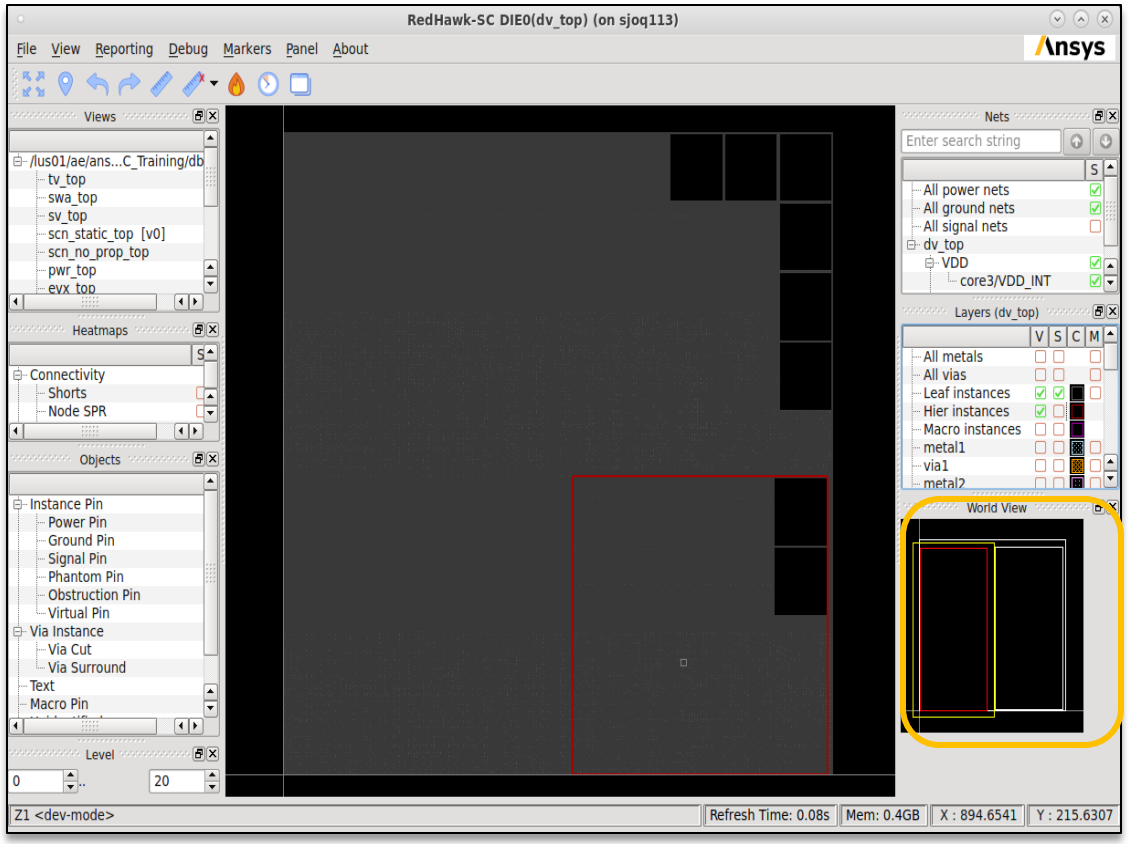
World View

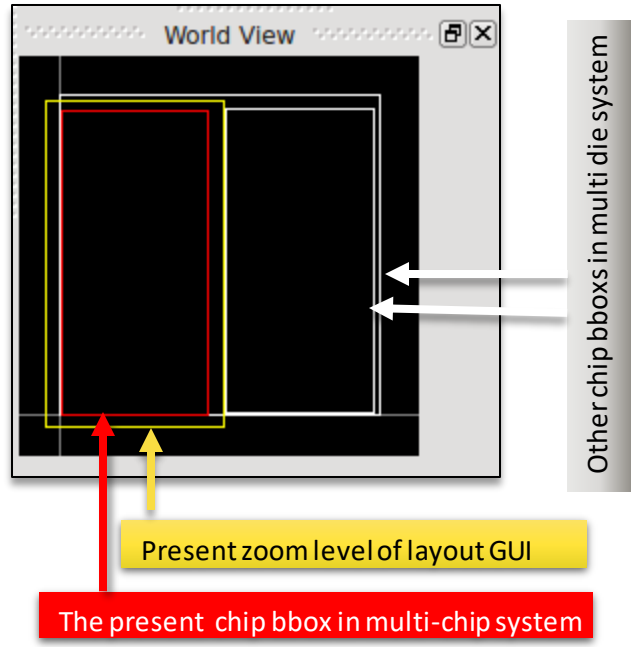
World view helps to understand relative position of a chip with respect to the multi-chip position.

This is a simplified layout of chip boundaries

Enabling world view:
Panel → check box on World View







Reporting Utilities

(EMIR reporting ,Userpython and General API)

3DIC Utility API's - Custom script

- This is a custom utility script for getting the multichip currents and voltages

Below is list of API's available in above custom script

```
get_multichip_total_bump_currents(multichip_analysis_view)

get_multichip_demand_currents(multichip_analysis_view)

get_multichip_bump_voltages_and_currents(multichip_analysis_view)

get_interface_currents(chip_analysis_view,multichip_config_view,interface)

get_interface_average_voltages(chip_analysis_view,multichip_config_view,interface)
```

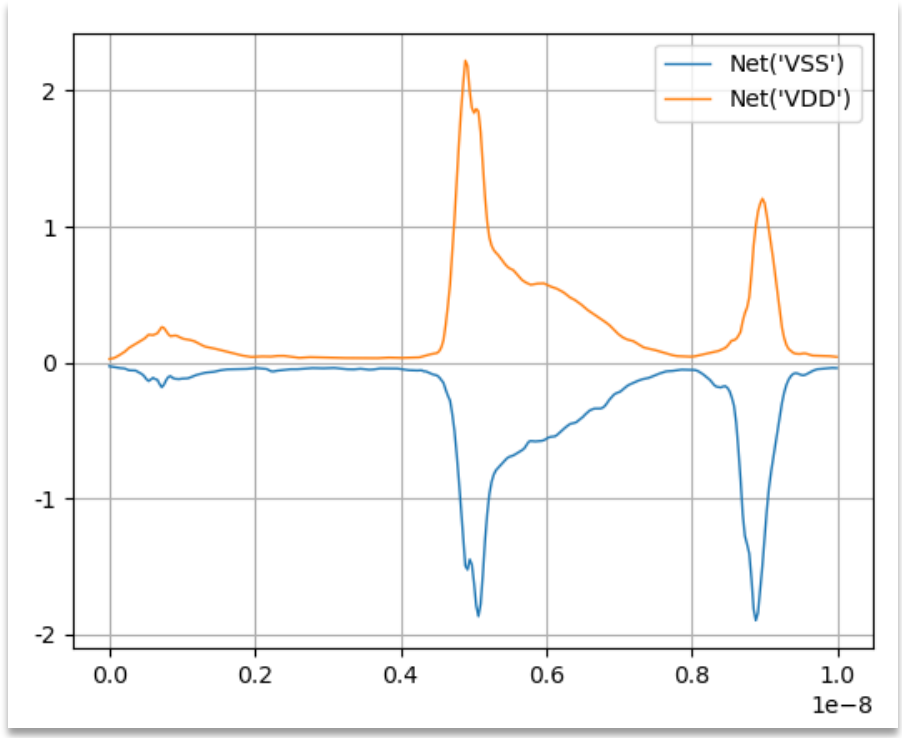
We need to include custom utility script 'multichip_utility.py'

System level bump currents

- Getting the total bump currents of the 3DIC system

```
Usage:
get_multichip_total_bump_currents(multichip_analysis_view)

   - It has a single argument – multichip analysis view
```

```
Example:
>>> total_bump_current =
get_multichip_total_bump_currents(av_3d)
>>> plot(total_bump_current)
```

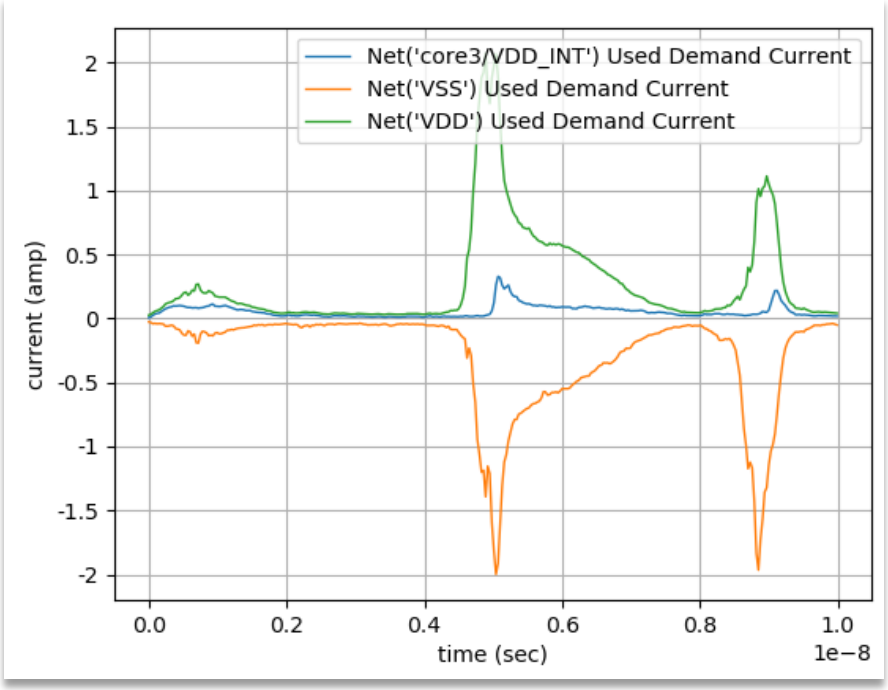


System level demand currents

- Getting the total demand currents of the 3DIC system

```
Usage:
get_multichip_demand_currents(multichip_analysis_view)

   - It has a single argument – multichip analysis view
```

```
Example:
>>> total_demand_current = get_multichip_demand_currents(av_3d)

>>> plot(total_demand_current)
```

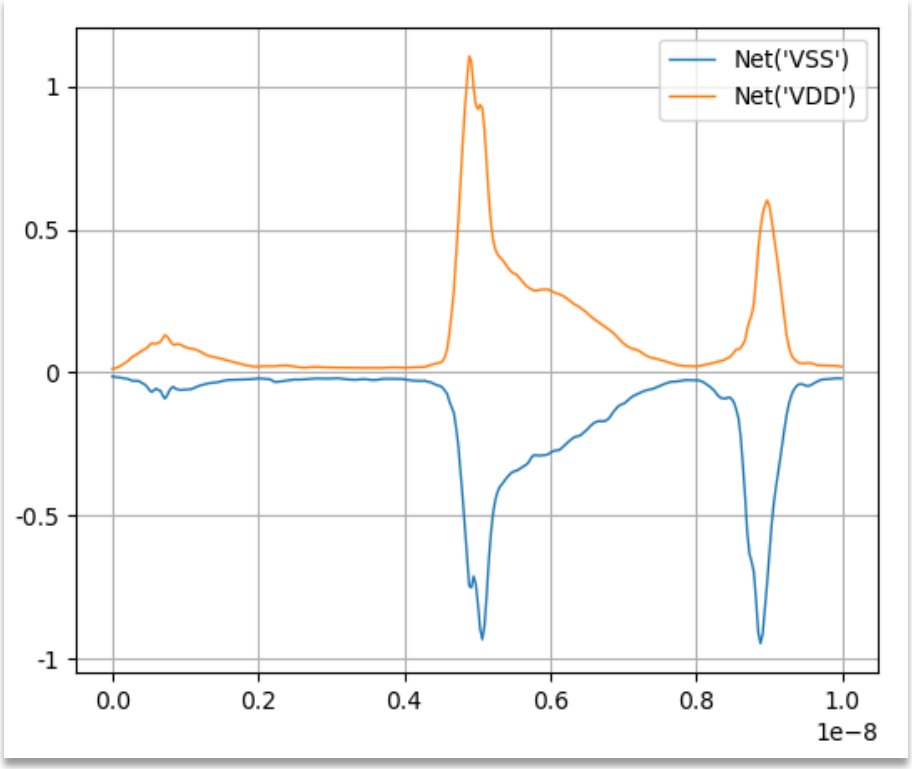


Bump waveforms of all interfaces

- Getting the currents and average voltages across every interface in the 3DIC system
- Each waveform stores in a dictionary

```
Usage:
get_multichip_bump_waveforms_per_interface(multichip_analysis_view)

   - It has a single argument – multichip analysis view
```

```
Example:
>>> get_multichip_bump_waveforms_per_interface(av_3d).keys()
['total_bump_currents', 'average_bump_voltages']

>>>get_multichip_bump_waveforms_per_interface(av_3d)['total_bump_cur
rents'].keys()
[Chip('DIE0'), Chip('DIE1'), Chip('Interposer')]
>>>
get_multichip_bump_waveforms_per_interface(av_3d)['total_bump_curren
ts'][Chip('DIE1')]
')][DieInterface(Die('Galaxy', 1), 'DIE_IF')]
>>> DIE0_total_bump_current =
get_multichip_bump_waveforms_per_interface(av_3d)['total_bump_curren
ts'][Chip('DIE1')][DieInterface(Die('Galaxy', 1), 'DIE_IF')]
>>> plot(DIE1_total_bump_current)
```

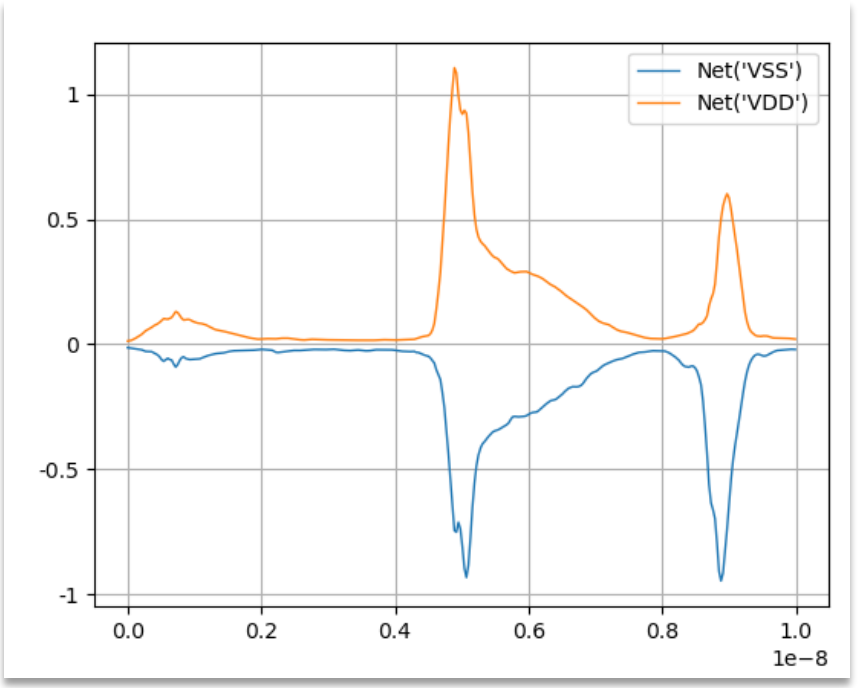


DIE interface currents

- Getting currents across the DIE interface specified by the user

```
Usage:
>>>get_interface_currents(chip_analysis_view,multichip_config_view,
interface)
    chip_analysis_view : Analysis View of the chip
    multichip_config_view : The MultiChipConfigView
    interface : interface object for which the user wants to
compute the currents across an interface
```

```
Example:
>>> av_DIE1 = av_3d.get_chip_view(Chip('DIE1'))
>>> DIE1_interface_cur =
get_interface_currents(av_DIE1,multichip_config,DieInterface(Die('G
alaxy', 1), 'DIE_IF'))

>>> plot(DIE1_interface_cur )
```

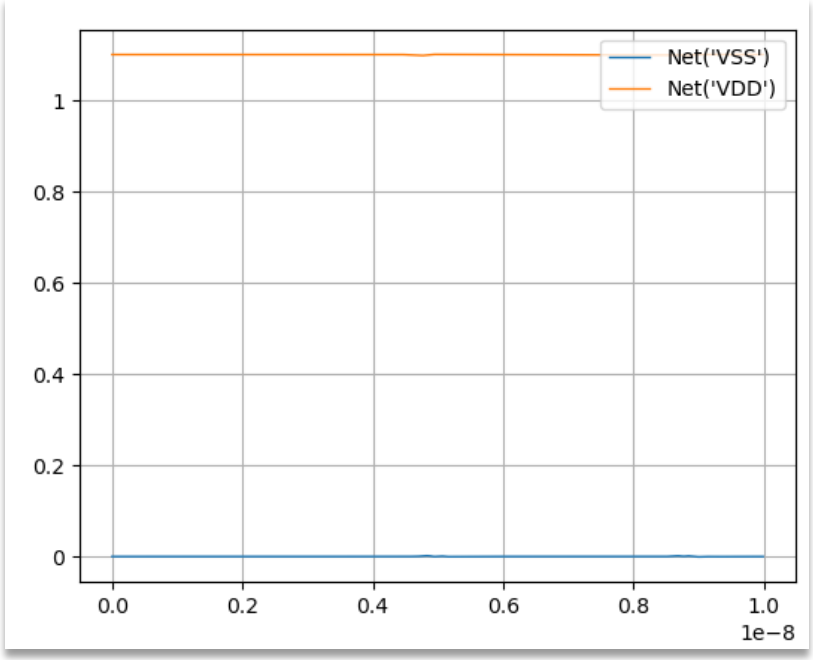


DIE interface average voltages

- Getting average voltages across the DIE interface specified by the user

```
Usage:
>>>get_interface_average_voltages(chip_analysis_view,multichip_conf
ig_view,interface)
    chip_analysis_view : Analysis View of the chip
    multichip_config_view : The MultiChipConfigView
    interface : interface object for which the user wants to
compute the voltages across an interface
```

```
Example:
>>> av_DIE1 = av_3d.get_chip_view(Chip('DIE1'))
>>> DIE1_interface_avg_vol =
get_interface_average_voltages(av_DIE1,multichip_config,DieInterfac
e(Die('Galaxy', 1), 'DIE_IF'))

>>> plot(DIE1_interface_avg_vol)
```



Multichip config view API's

- Some of useful multichip config view API's and also useful for custom scripts

```
>>> multichip_config.get_chips()
[Chip('Interposer'), Chip('DIE0'), Chip('DIE1')]

>>> multichip_config.get_dies()
[Die('Galaxy', 1), Die('GalaxyInterposer', 2)]

>>> multichip_config.get_package_connections()
[(Chip('Interposer'), DieInterface(Die('GalaxyInterposer', 2), 'IF_PKG'))]

>>> mccfg.get_inter_chip_connection_parasitics()
{'CP_1': {'capacitance': 1.0000000036274937e-15,
'height': 5.0,
'inductance': 9.999999960041972e-13,
'length': 20.0,
'resistance': 9.999999747378752e-05,
'width': 20.0},
'RHSCDEFAULTCP': {'capacitance': 0.0,
'height': 0.0,
'inductance': 0.0,
'length': 1.0,
'resistance': 9.999999974752427e-07,
'width': 1.0}}
```

Multichip analysis view API's

- Getting multichip info from analysis view

  av_3d.get_multichip_info()

```
Example:
>>> av_3d.get_multichip_info().keys()
['DIE0', 'DIE1', 'Interposer']
>>> av_3d.get_multichip_info()['DIE0']
{'dv': <gp.ModifiedDesignView object at 0x7f8f0ac4eb50>, 'chip_id': 0, 'angle': 0.0,
'flip': False, 'coord': RealCoord(22.000000,0.000000)}
>>> av_3d.get_multichip_info()['DIE1']
{'dv': <gp.ModifiedDesignView object at 0x7f8f0ac4eb50>, 'chip_id': 1, 'angle': 180.0,
'flip': True, 'coord': RealCoord(2652.829956,12.999900)}
>>> av_3d.get_multichip_info()['Interposer']
{'dv': <gp.ModifiedDesignView object at 0x7f8f0ac541d0>, 'chip_id': 2, 'angle': 0.0,
'flip': False, 'coord': RealCoord(0.000000,0.000000)}
```

Multichip analysis view API's

- User can access the individual analysis view of each chip using below command

```
av_3d.get_chip_view(Chip('chip_name'))
```

```
Example:
>>>av_memory = av_3d.get_chip_view(Chip('memory'))
>>>av_asic = av_3d.get_chip_view(Chip('asic'))
>>>av_interposer = av_3d.get_chip_view(Chip('interposer')
```

- Above analysis views are same as single chip analysis views but those have an effect of multichip analysis in it

- All the APIs are same as in single chip analysis views , so there is no special APIs in multichip analysis view , we need get the results from individual chips analysis view

Multichip EMIR Reporting

- User can access the EMIR reports from individual analysis view of each chip

```
av_3d.get_chip_view(Chip('chip_name'))
```

```
Example:
>>>av_memory = av_3d.get_chip_view(Chip('memory'))
>>>av_asic = av_3d.get_chip_view(Chip('asic'))
>>>av_interposer = av_3d.get_chip_view(Chip('interposer')
```

- There is no direct EMIR reporting from multichip analysis view , We need get individual chip analysis views from multichip analysis view and get EMIR reports

```
emir_reports.write_all_instance_voltages(av_asic)

emir_reports.write_em_metal_report(em_asic)
```

Thank You

Ansys