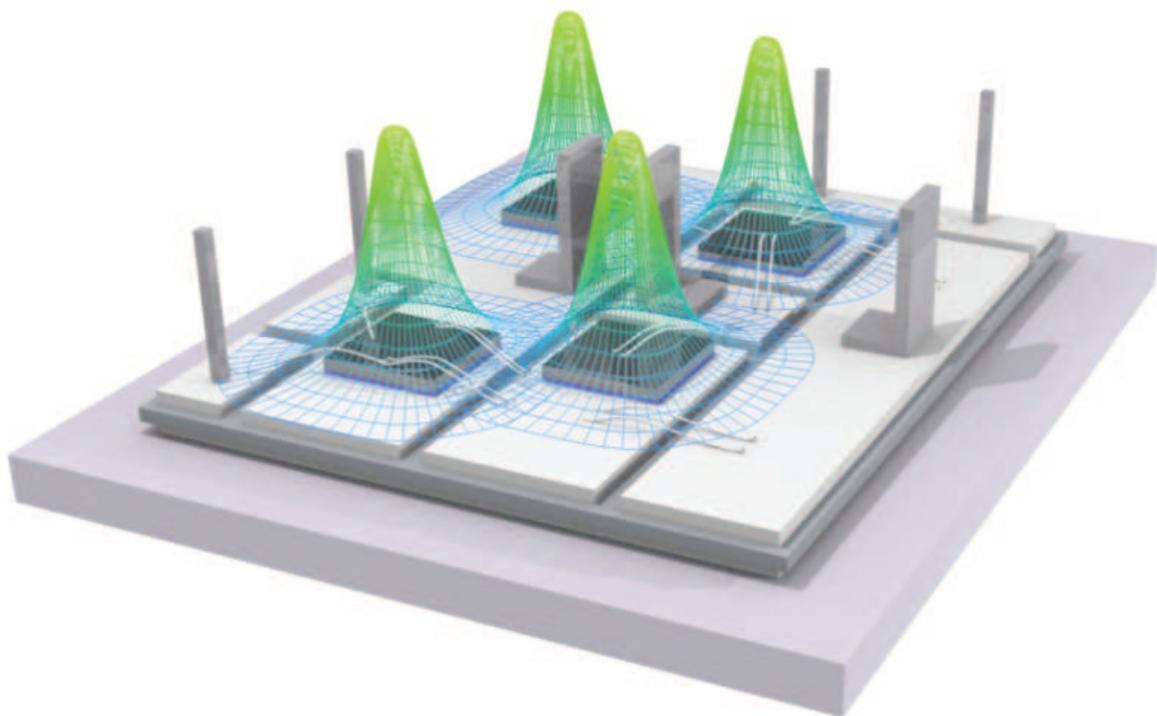




Mixed-Signal Computer-Aided Design (MSCAD) Laboratory
Energy-Efficient Electronics and Design Automation (E³DA) Laboratory

PowerSynth User Manual Version 2.1



Contents

1	Introduction	2
1.1	Executive Summary	2
1.1.1	PowerSynth Introduction	2
1.1.2	About Version 2.1	2
1.2	Organization	3
1.3	PowerSynth 2 Architecture	3
1.4	User Interfaces and Design Input	3
1.5	Layout Engine	4
1.6	Performance Evaluation Models	6
1.7	Design Optimization and Solution Export	6
2	Using PowerSynth v2.1	7
2.1	Installation and Usage	7
2.2	Requirements	7
2.2.1	Technology Library Content	7
2.2.2	Initial Layout Description	11
2.2.3	Constraints	20
2.3	PowerSynth 2.0 GUI Introduction	22
2.4	Walk-Through Three Examples	32
2.4.1	2D MCPM Layout Optimization	32
2.4.2	3D MCPM Layout Optimization	44
2.4.3	Converter Layout Optimization (ongoing work)	49
3	PowerSynth-Related Publications	52
3.1	Useful Links	54
4	Authors	54
4.1	Graduate Research Assistants/ Graduates	55
4.2	Supervisors	55

1 Introduction

1.1 Executive Summary

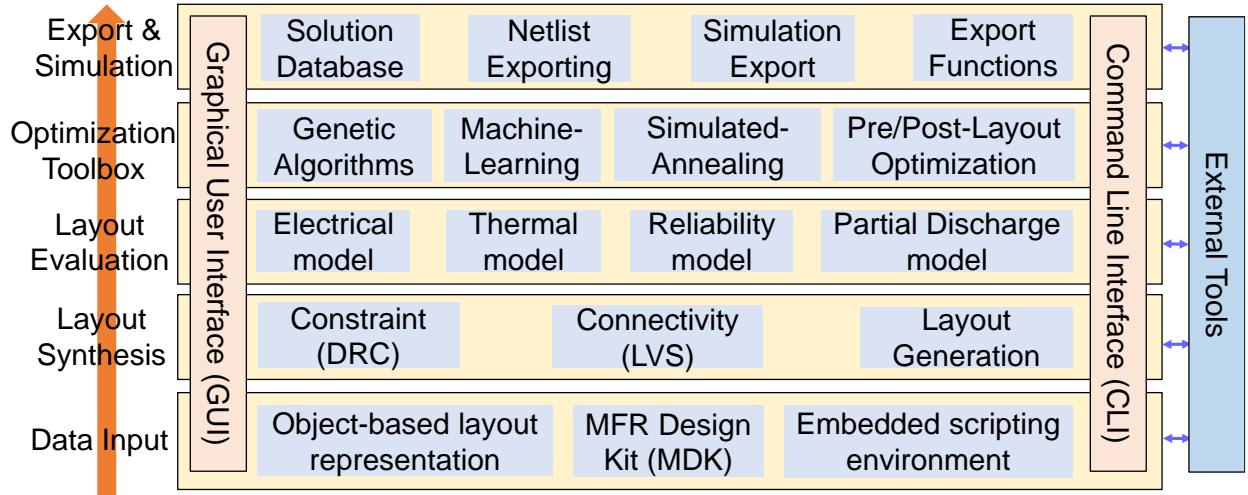
1.1.1 PowerSynth Introduction

PowerSynth is an electronic design automation (EDA) tool that can synthesize and optimize multi-chip power module (MCPM) layouts with significantly faster than any commercial tools. PowerSynth currently performs multi-objective optimization to produce Pareto-front solutions to the proper placement of power semiconductor device die and the routing of metal traces on ceramic substrates. The tool accounts for temperature distributions and electrical parasitics as a function of the layout geometries that it considers. This tool has been hardware-validated. Continued research on this project will further elaborate the capabilities by extending the work to greater fidelity in thermal, electrical, and mechanical domains.

1.1.2 About Version 2.1

As a part of PowerSynth continuous research and development, this version (v2.1) offers a command line interface and graphical user interface (GUI) to both Linux and Windows machines to the users with a significant improvement in layout representation and generation methodology. Some of the major features in this version include:

- Implement hierarchical optimization for layout generation (version 2.1).
- Add Multi-Objective Particle Swarm Optimization (MOPSO) as a new optimization algorithm for layout generation (version 2.1).
- Extended layout handling from module to converter design (version 2.1). It is noted that the converter layout design is in the experimental phase.
- New modular and hierarchical software architecture for PowerSynth 2.0 (shown in Figure 1).
- Both graphical user interface (GUI) and command-line interface (CLI) for the user's flexibility.
- Generic layout representation technique for complex (2D/2.5D/3D) geometry handling.
- Hierarchical constraint-aware, generic, efficient, and scalable layout generation methodology.
- Heretogeneous components handling.
- Both intra-layer (wire bond) and inter-layer (via) interconnects handling capability.
- Voltage-current dependent intra-layer (2D) reliability constraints handling.
- Arbitrary number of passive layers in the layer stack.
- Initial(single) layout performance evaluation.
- Hardware-validated partial element equivalent circuit (PEEC) based built-in model for 2D/2.5D layouts and FastHenry for 3D layouts electrical performance evaluation.
- Hardware-validated, ParaPower thermal model.
- Optimization algorithm options: randomization and genetic algorithm (NSGAII).
- Most of the state-of-the-art packaging technology (2D/2.5D/3D wire-bonded, wire bondless, hybrid) can be considered for optimization.



Design Flow PowerSynth 2 Core: 2D/2.5D/3D Designs, Python 3.10, QT 6.5, MATLAB 2022b, Win/Linux

Figure 1: PowerSynth 2 architecture

1.2 Organization

After a brief introduction to PowerSynth 2 architecture, this document introduces the GUI workflow. This document will show the user how to prepare the necessary files, and parameters for using this version. Then, this document walks the user through the steps to optimize a sample 2D half-bridge power module, a 3D half-bridge power module, and a converter design.

1.3 PowerSynth 2 Architecture

PowerSynth 2 architecture is shown in Figure 1. Compared to PowerSynth 1, the new architecture is a module-based one. It has multiple design layers, and each layer has a flexible API to communicate within or outside the tool. The tool has two fundamental parts: A core that contains the built-in algorithms, methodologies, and modeling techniques; External tools that include commercial tools or models developed by other research groups, which are linked through APIs developed by the authors. This architecture is scalable and can be extended toward cabinet-level optimization.

1.4 User Interfaces and Design Input

PowerSynth v2.1 has both GUI-based interactive mode and CLI-based unattended mode to support Windows and Linux compatibility. The command-line interface works on user input through the terminal, which uses a macro script describing the necessary parameters to perform the optimization flow. Besides, there is an interactive GUI for the users. The main window of the user interface provides two main flows: (a) Creating a new project from an existing layout; (b) Running a project using the macro script already prepared. Users can modify the material library, layer stack, and design constraints through the GUI. Then, the layout generation/optimization setup window can run PowerSynth 2 in three different modes: performing performance evaluation of the initial layout, generating layout solutions only, and optimizing layout solutions based on the performance models. For optimization/evaluation mode, the user needs to set up necessary files and parameters through different model setup windows. Once the necessary parameters are defined, the layout optimization process begins, and results are shown in the solution browser. Finally, the user can choose an

individual solution from the solution space and export both individual and complete solution space for further processing.

PowerSynth 2 requires an initial description of the technology that contains the layer stack, power devices, substrates, connectors, heat spreaders, wire bonds, and via information. As an umbrella module, the built-in manufacturer design kit (MDK) contains a library of materials and other technology information similar to the process design kit (PDK) in Very Large-Scale Integration (VLSI). An interface is built to interact with MDK so that users can update the libraries. Besides, the initial placement of the devices, leads, and routing of the traces information is taken through a hierarchical geometry description script, which is represented through an object-based data structure. An embedded scripting environment can be used to accelerate the layout geometry processing.

1.5 Layout Engine

The significant improvements with the hierarchical constraint-aware layout engine are:

- An interactive constraint input feature which is helpful for user to specify or modify design constraint values to have different layout structures.
- Three types of layout generation capability: minimum-sized layout, variable floorplan sized, and fixed floorplan sized.
- As the engine takes into account of all design constraints in the layout generation phase, it always generates 100% manufacture-able solutions.
- The updated layout engine can incorporate different types of optimization algorithms (i.e., genetic algorithm, particle swarm, randomization)
- This layout engine treats each component as rectangle, so geometrical complexity is not a problem.
- This engine can process broader range of layouts even considering heterogeneous components (e.g. passive components, sensors, etc.).
- As the updated layout engine is constraint-aware, different types of constraints can be declared : design constraints, reliability constraints, user-defined constraints. Generated solutions always satisfy all the given constraints.
- Scalable, efficient algorithms have a linear time complexity irrespective of layout types (2D/2.5D/3D).
- 2D/2.5D/3D layouts with most of the state-of-the-art (SOTA) packaging technologies (wire-bonded, wire bondless, hybrid, etc.) can be optimized using the updated algorithms.

Methodology

From the user-defined initial input script, using corner stitch data structure (used in Magic VLSI tool), a collection of rectangular tiles are stored in a hierarchical tree structure. Based on design constraints, constraint graphs (popular in VLSI floorplan compaction) are created for each corner-stitched plane. Two types of constraint graphs are considered: horizontal constraint graph (HCG) and vertical constraint graph (VCG) for maintaining horizontal and vertical relationship among components. These constraints are evaluated using the longest path algorithm and the results are propagated through the tree. Bottom-up constraint propagation and top-down location propagation algorithms are implemented to generate solution. PowerSynth previous versions are restricted

with handling 2D/2.5D layouts only. Therefore, the constraint graphs creation are performed by one-to-one mapping with the corner-stitching tree structure. However, in PowerSynth 2 inter-layer connectivity has been considered by introducing two types of via structures: through DBC type and port type. To handle the constraints across different layers, abstract nodes are generated in the hierarchy tree, which do not have any corner stitch plane but have constraint graphs. Interfacing layers are introduced for handling these via type connections, which have been the key updates from previous version (v1.9). Detail algorithms can be found in. Some concepts associated with layout generation are described as follows:

- **Constraints:** Two types of constraints are considered: (a) design constraints, (b) reliability constraints.
 - (a) **Design Constraints:** These are standard design rules from the manufacturer. Three types of design constraints are considered.
 - Dimension Constraints:** Here, minimum width along x-axis (Min Width), minimum width along y-axis (Min Height) and minimum enclosure (Min Enclosure) are specified for each type of component.
 - Spacing Constraints:** In this table, minimum spacing values between every pair of components are declared.
 - Enclosure Constraints:** When a component is placed on top of another component, there may be some minimum enclosure value. So, this table has all possible minimum enclosure values.
 - (b) **Reliability Constraints:** These constraints are user-defined based on the high-voltage-current applications. To minimize partial discharge phenomena, and increase the reliability of the power module, user can define voltage-dependent minimum spacing and current-dependent minimum width constraints.
- **Operating Modes:** Based on the evaluation of the constraint graphs, there are three modes of operation (shown in Table 1).

Table 1: Summary of operating modes

Mode	Purpose	Evaluation Methodology
0	Minimum sized layout	Minimum constraint values
1	Variable floorplan layouts	All Weights are randomized with minimum constraints. No maximum constraints
2	Fixed floorplan layouts	All Weights are randomized with minimum constraints. Some have maximum constraints

- **Minimum Size Layout:** This layout is generated using all minimum constraint values. So, this layout reflects maximum possible power density for a layout. As this is the minimum sized solution, it is electrically optimized but thermal performance is so poor.
- **Variable Size Layout:** If this mode is selected, all constraint values are randomized and new layout solution is generated. User can generate arbitrary number of valid layout solutions with different floorplan size.
- **Fixed Size Layout:** All edge weights are randomized within given area to generate arbitrary number of solutions. As floorplan size is always fixed there is less variation in this mode than the previous one.

1.6 Performance Evaluation Models

As WBG devices can switch faster at higher voltage and current, electrical parasitics in the MCPM layout must be minimized to achieve the target circuit performance. With the increased density in a 3D MCPM layout design, the parasitic loop inductance is significantly reduced compared to its 2D counterparts. However, as the 3D layout solution is more compact, ensuring thermal and mechanical reliability becomes challenging. Therefore, electro-thermo-mechanical performance and reliability optimization are required before fabricating a module. Available multi-physics or FEA-based analysis tools can be used for capturing these performances. However, these methods are not efficient to be used in the optimization loop due to long runtime. To address these issues, PowerSynth 2 is equipped with reduced-order electrical and thermal models, which are fast and quite accurate compared to FEA tools. The electrical model performs resistance, capacitance, and inductance evaluation. Multi-level APIs have been developed inside the tool to leverage the existing electrical, thermal, and mechanical models from other research groups or companies. In this version, hardware-validated PEEC model and FastHenry from FastFieldSolvers is used for loop parasitics extraction of 2D/2.5D layouts, and 3D layouts, respectively. And ParaPower from Army Research Lab has been enabled for thermal evaluation.

1.7 Design Optimization and Solution Export

For providing optimization options, PowerSynth 2 has a genetic algorithm (NSGAII) and built-in randomization algorithm framework. Currently, PowerSynth 2 flow users can choose any available options for performing electro-thermal optimization. A comparison study between randomization and genetic algorithm shows that genetic algorithm can converge faster to the Pareto-optimal solution set for a given number of generations. Though randomization provides little guidance toward optimization objectives, it can explore a larger solution space and potentially find better solutions with an acceptable runtime overhead. Once the optimizer generates the solution space, a non-dominated sorting is applied to get the Pareto-optimal solutions. In this version, along with the Pareto-front solutions, an entire solution space is also reported after optimization. Moreover, for each solution, the layout geometry is also exposed in a CSV file containing each component's coordinates, width, and length. This information helps a designer to regenerate the geometry script for the solution layout. Exporting a complete distributed parasitic netlist with RLC elements is one of the killer features. The exported netlist can back-annotate the circuit schematic to perform re-simulation, completing the round-trip engineering design loop before fabrication. Finally, the optimized solution can be fabricated to validate and fine-tune models through physical measurements.

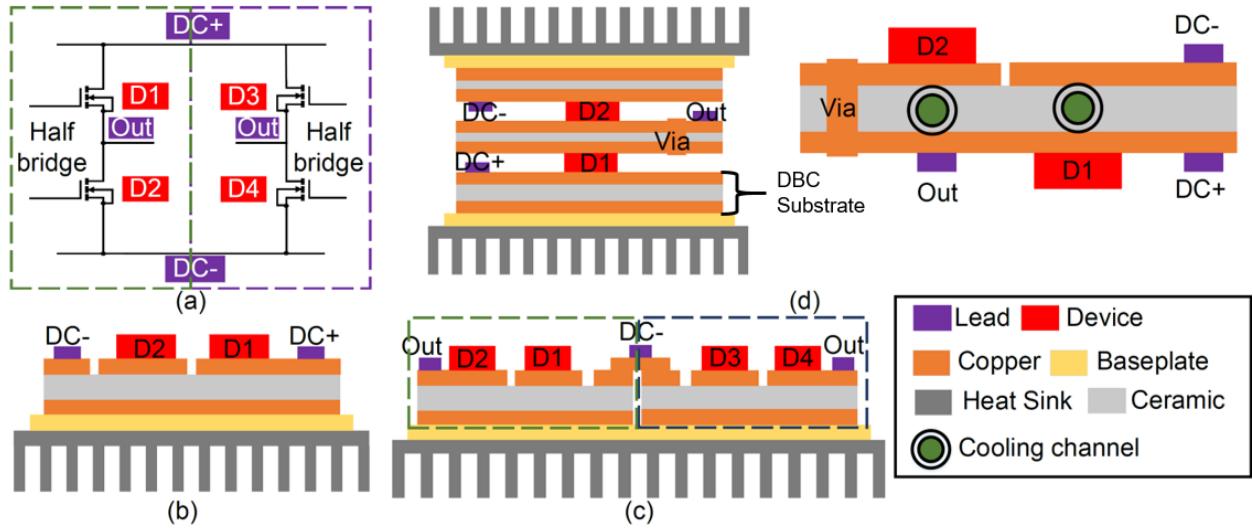


Figure 2: (a) Circuit schematic of power module, Various MCPM layout structures: (b) 2D half-bridge, (c) 2.5D full-bridge, and (d) 3D with double-sided cooling or embedded cooling

2 Using PowerSynth v2.1

2.1 Installation and Usage

PowerSynth v2.1 can be installed on Linux and Windows machines. PowerSynth v2.1 requires Matlab to run the ParaPower thermal model. The package uses Python3.10 and assumes Matlab 2022b is installed at the default location. More details about the installation of each package can be found at <https://github.com/e3da/PowerSynth2-pkg>.

For Windows:

After installation, PowerSynth2-GUI and PowerSynth2-CLI shortcuts are available to use. To use GUI click on PowerSynth2-GUI and to use CLI after clicking on PowerSynth2-CLI use this command to run the PowerSynth:

```
.\python .\pkg\bin\PowerSynth2.py .\pkg\work\Sample_Designs\2D_Case_3\macro_script.txt
```

For Linux:

After installation, to use GUI inter PowerSynth command, and to use CLI, first use this command to set the environment:

```
setenv PATH ./PowerSynth2.0/bin:$PATH
```

Then, use the below command to run the PowerSynth:

```
PowerSynth2 ./PowerSynth2.0/pkg/work/Sample_Designs/2D_Case_3/macro_script.txt
```

2.2 Requirements

2.2.1 Technology Library Content

To design a standard 2D/2.5D/3D power module (shown in Figure 2), the key elements are as follows:

1. Baseplate
2. Plain (for 2D/2.5D) or Through ceramic via connected (for 3D) Substrate (Direct Bonded Copper (DBC)): Back-side metal, Ceramic, Top-side metal)
3. Components (Devices: MOSFETs, Diodes, IGBTs, Capacitors, etc.)
4. Connectors (Leads: power and signal)
5. Bonding wires, metallic posts (via).

To describe the structure and layout of the module, the required information are taken through several files. The files associated with these elements are as follows:

1. Layer stack (.csv file)
2. Parts (.part file)
3. Wires (.wire file)

Each file content is described below:

1. **Layer stack:** This file provides the dimensions, material information about baseplate, and substrate. These information are taken input as a CSV file.

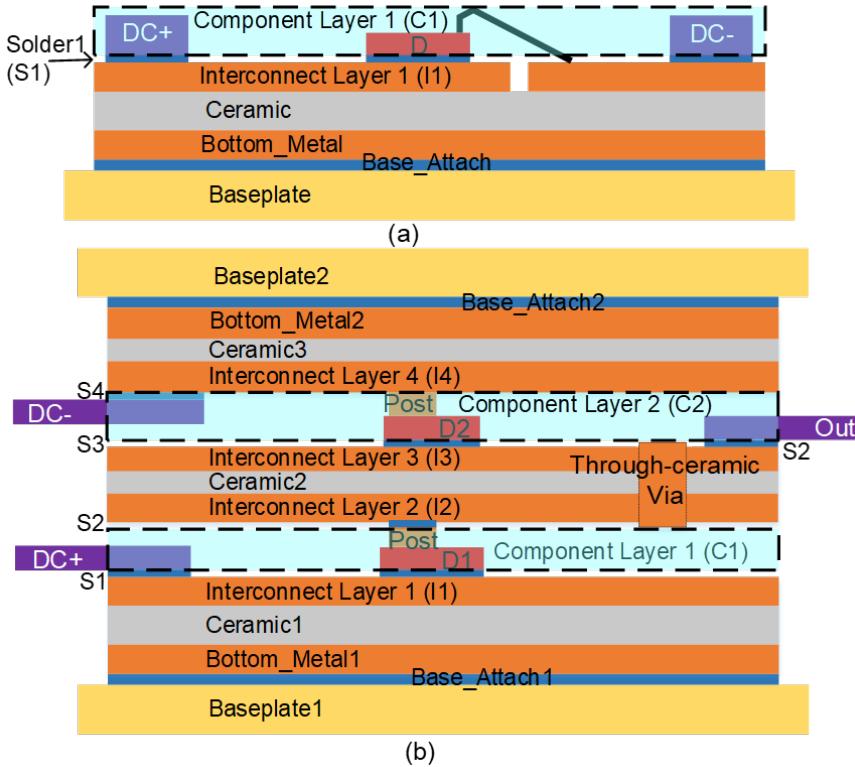


Figure 3: Sample layer stack of a (a) 3D module, (b) 2D module

A sample 2D and 3D layer stack are shown in Figure 3(a), and (b), respectively. The 3D layer stack file contents are shown in Table 2. Any layer stack file has nine columns:

1. ID: An integer to uniquely identify each layer.
2. Name: Each layer needs to have a name (i.e., B1 for Baseplate layer 1, M1 for substrate backside metal 1, D1 for dielectric layer 1 of the substrate, I1 for interconnect layer 1 of the substrate, and C1 for component layer 1). Since this version supports multi-layer stacked DBC, all layers except the component layer can be multiple.
3. Origin: Defines reference X and Y coordinate as Z coordinate is always considered to be

- starting at 0. All units are in mm.
4. Width: Defines width of each layer in mm.
 5. Length: Defines length of each layer in mm.
 6. Thickness: Defines thickness of each layer in mm.
 7. Material: Name of the corresponding layer material (needs to be same as in material library).
 8. Type: Two types are allowed: p for passive and a for active. Only component layer is an active layer and rest of the layers are passive.
 9. Electrical: This field is for electrical performance evaluation. Here, F for floating, G for ground, D for dielectric, S for signal, and C for component.

More examples are available in the ‘**Sample_Designs**’ directory of the package.

Table 2: Content in a layer stack file

ID	Name	Origin	Width	Length	Thickness	Material	Type	Electrical
1	Baseplate1	0,0	44	28	1	copper	p	F
2	Base_Attach1	5,5	34	18	0.1	SAC405	p	D
3	Bottom_Metal1	5,5	34	18	0.2	copper	p	G
4	Ceramic1	5,5	34	18	0.64	Al_N	p	D
5	I1	5,5	34	18	0.2	copper	p	S
6	S1	5,5	34	18	0.1	SAC405	p	D
7	C1		34	18	2	None	a	C
8	S2	5,5	34	18	0.1	SAC405	p	D
9	I2	5,5	34	18	0.2	copper	p	S
10	Ceramic2	5,5	34	18	0.32	Al_N	p	D
11	I3	5,5	34	18	0.2	copper	p	S
12	S3	5,5	34	18	0.1	SAC405	p	D
13	C2		34	18	2	None	a	C
14	S4	5,5	34	18	0.1	SAC405	p	D
15	I4	5,5	34	18	0.2	copper	p	S
16	Ceramic3	5,5	34	18	0.64	Al_N	p	D
17	Bottom_Metal2	5,5	34	18	0.2	copper	p	G
18	Base_Attach2	5,5	34	18	0.1	SAC405	p	D
19	Baseplate2	0,0	44	28	1	copper	p	F

While creating a 3D layer stack, the component layer (C) thickness needs to be set carefully as this layer has all the leads, devices, vias (posts), etc. The clearance of these components need to be aligned with the thickness of the layer. However, in the case of 2D layer stack, component layer is the top most layer and so the thickness of that layer is optional. For both of the cases, the material information of the component layer is not required in the layer stack as that information is read from individual component’s description file.

2. **Parts:** The components (active and passive devices), connectors (leads, vias) are considered as parts in this version. The dimensions, material information of these elements are taken from corresponding .part file. These files are written in a text editor and saved as .part extension. A sample MOSFET.part and power_lead.part file content are shown in Figure 4 (a), (b), respectively.

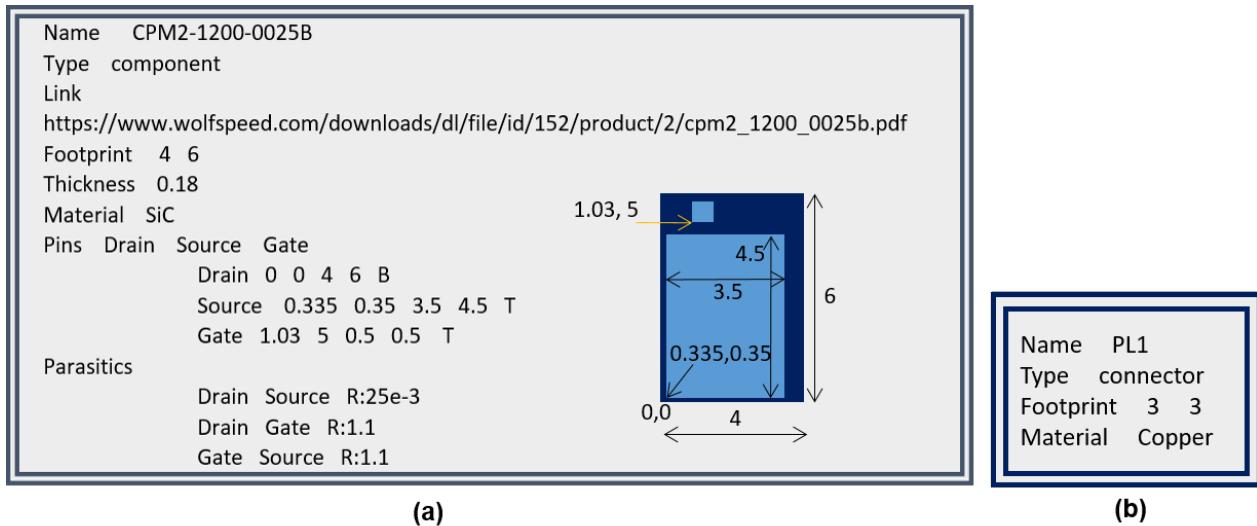


Figure 4: Content in (a) MOSFET.part, (b) power_lead.part file

In Figure 4 (a), each row has a key name e.g. “Name” and a value separated by a space e.g.: “CPM2-1200-0025B”. The definition for each key is as follows:

Name: Name of the Part

Type: There are 2 options: component (device like MOS, diode, capacitor, etc.) or connector (terminals like power lead, signal lead, via etc.)

Link: A link to the part datasheet

Footprint: Width <space> Length (Provide the footprint of the component in mm)

Thickness: Thickness of the component (in mm)

Material: Name of the Material (e.g., SiC) [should match with material library from MDK (materials.csv file in MATERIAL_LIB_PATH of "settings.info" file.)]

Pins [list of preferred pin names separated by space]: Provide a list of pin names

Pin_name: For each pin provide a pin pad rectangle (bottom-left coordinate x <space> y <space> width <space> height) reference to the component bottom left corner. At the end of the pin rectangle add a keyword B or T to distinguish between Bottom and Top side pins.

Parasitics: Used to provide component internal parasitic information.

Pin_name1 <space> Pin_name2 <space> R: R_val <space> L:L_val <space> C:C_val

Provide a list of RLC value between every 2 pins (with internal parasitics)

For different devices, the keys need to be same, but the values will be changed depending on the device type. In Figure 4 (b), each row has a key, value pair separated by a space. The definitions for each key are similar to those in Figure 4 (a).

All .part files need to be saved in the "Part_Lib" folder. More sample .part files are provided in the ‘Sample_Designs’ folder inside ‘Part_Lib’.

3. **Wires:** The wire standard, resistivity, and radius information are stored in a *.wire file. The file content are written in a text editor and saved as .wire extension. Content of a sample wire file is shown in Figure 5. In the *.wire file, the first line is the wire bonding standard. This will affect the parasitic extraction of the bond wire group. The second line is the resistivity of the material. This is used to compute the parasitic resistance of the wire (unit is Ωm). The third line is used to provide the radius of the wire in mm. For bond wire with square cross

section, the effective radius can be used. In the last two lines, there is a space in between key and value. All *.wire files need to be saved in ‘**Wire_Lib**’ folder.

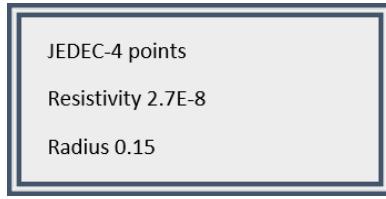


Figure 5: Content in a *.wire file

2.2.2 Initial Layout Description

Once the technology library related files are ready, the initial layout needs to be described through the text (.txt) files. PowerSynth 2 supports two formats for declaring the initial layout: (a) Full-Custom Mode Script that gives freedom for custom placement of all components including the bonding wires and all types of vias; (b) Semi-Custom Mode Script that makes the layout description easier and finds the wire bond/post-type via locations automatically by imposing some restrictions on the component usage. User can choose any of these two formats. Both modes are described below.

(a) Full-Custom Mode Script: This mode is useful to declare the initial layout with full freedom. All previous versions of PowerSynth use this mode. In this mode, user needs to draft every element locations carefully by using the global coordinate system for the entire layout. This might be a tedious task for some complex layouts. On the other hand, the pros of this format is the user can try any Manhattan layouts that do not need to be always hardware feasible with the state-of-the-art manufacturing technologies. This mode is helpful for the researchers and developers, who want to study new types of custom layouts. Two files are required to describe an initial layout of a power module in this mode:

1. Layout geometry description script
2. Connectivity setup script.

Each of the file content is described below in detail:

1. **Layout geometry description script:** This script is a text (.txt) file to input the initial layout geometry of the layout. This script contains the initial placement and routing of the components, and connectors. A sample 2D, and 3D layout script file content are shown in Figure 6, Figure 7, respectively.

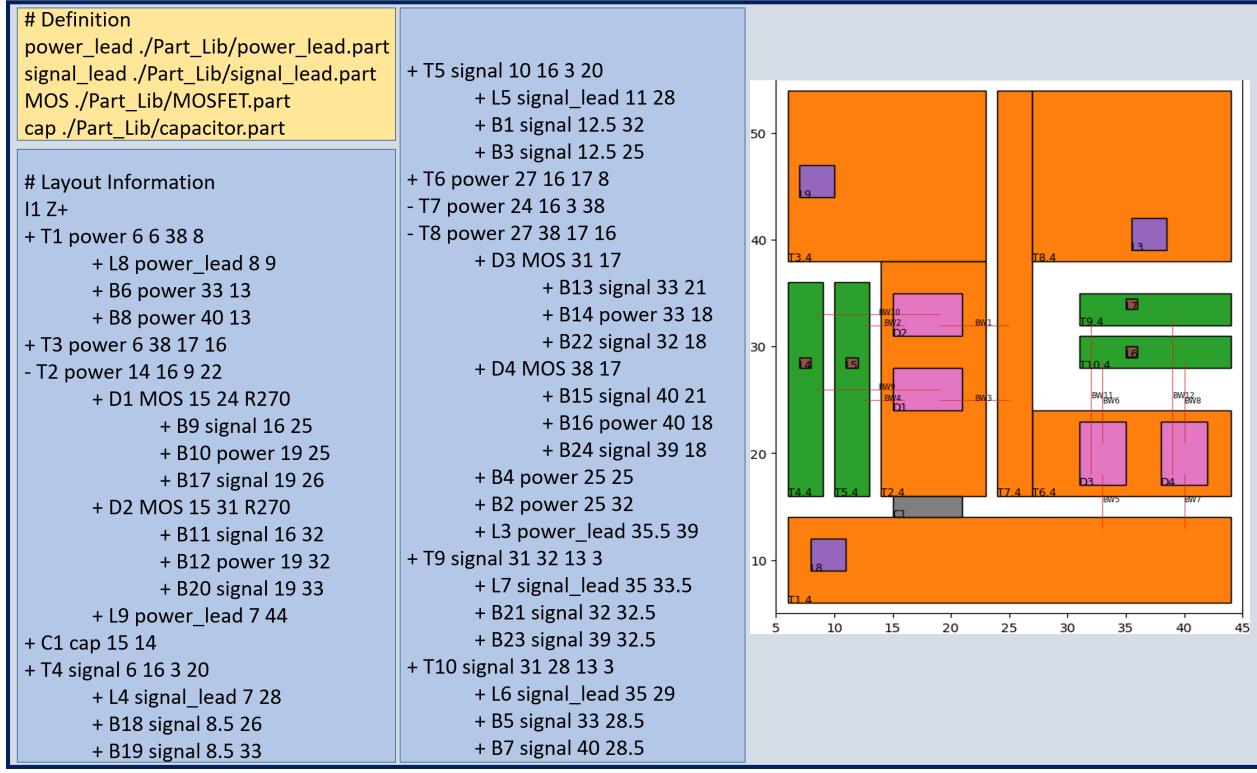


Figure 6: Content in a full-custom 2D layout geometry description script

The file has two (2D layout) or three (3D layout) sections:

(a) Definition

In this part, the necessary parts (.part) file locations are provided. Anything below the tag “#Definition” and up to the tag “#Layout Information” in the script are considered in this section. For any new test case, the tags should not be changed. Also, a blank line is required in between two sections (as shown in Figure 6).

The user can use any keywords to represent a component or connector which is defined by the *.part file. These keywords need to be reused in the “#Layout Information” section. In the “#Definition” section, in each line, there are two segments:

<segment1> <space> <segment2>.

<segment1> is the keyword of the component or connector (MOS, Diode, IGBT, power_lead, signal_lead, Via, etc.)

<segment2> is the relative location of the component.part file.

(b) Via Connectivity Information

This section is required for 3D layout geometry description. In this part, all via type connections are declared using four fields separated by a space:

1. Layout component id (V1: Via 1 or V2: Via 2, etc.)
2. Via connecting routing layer 1 (I1: Interconnect layer 1 or I2: Interconnect layer 2, etc.)
3. Via connecting routing layer 2 (I1: Interconnect layer 2 or I2: Interconnect layer 3, etc.)
4. Via type ('Through' or 'Port'). 'Through' type via is required explicit declaration in the script as shown in Figure 7. 'Through' via refers to the vias those are between two routing layers of the same substrate (through ceramic via). The other type of via connects two routing layers of two different substrates and known as 'Port' type vias. However, if the via type is

not mentioned in the script by default it is ‘Port’ type. So, in the layout description script, there is no need to mention the type (4th field) of such vias. These vias represent the metallic post-type connections as well.

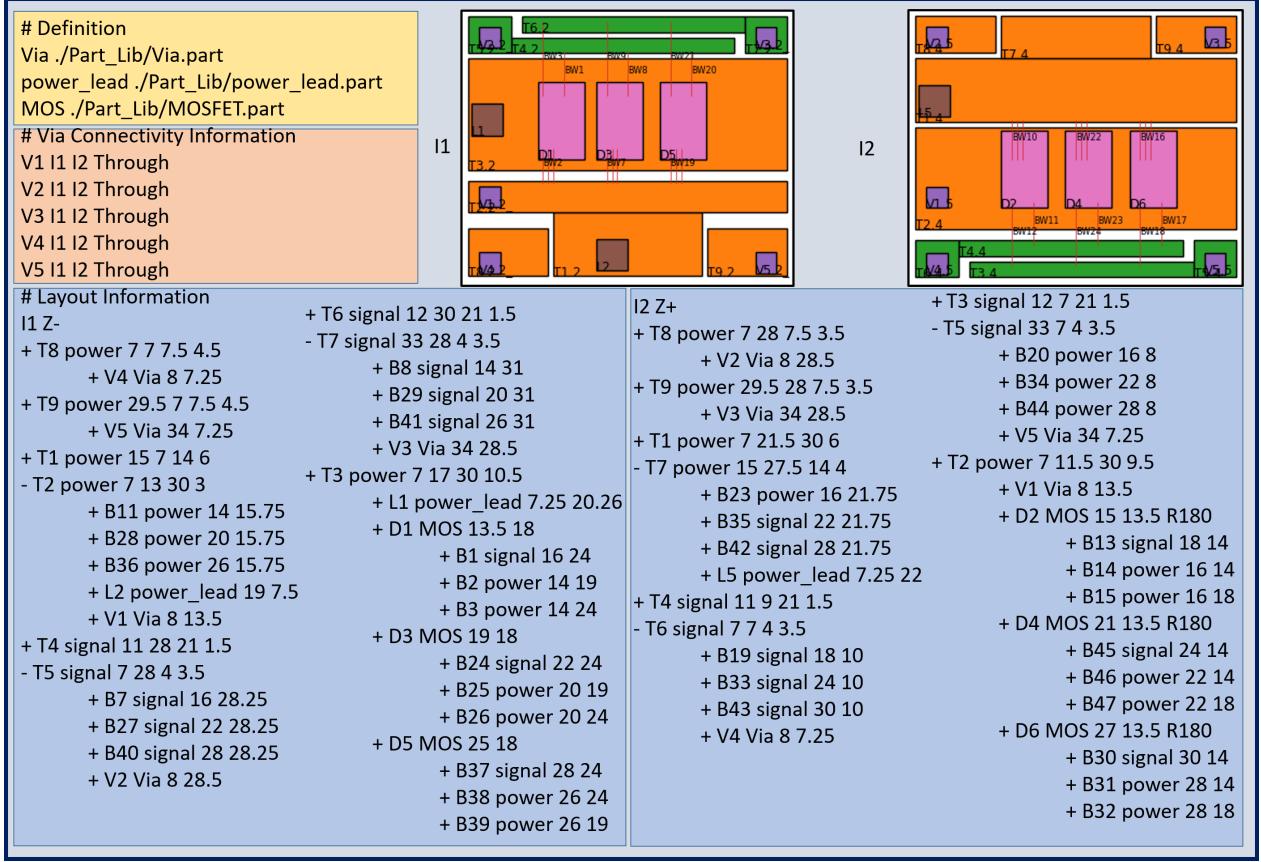


Figure 7: Content in a full-custom 3D layout geometry description script

(c) Layout Information

This section describes the layout geometry and component hierarchy information for the layout engine. Anything below the tag “#Layout Information” are used to describe the layout geometry. The geometry should be described hierarchically and the hierarchical order is bottom-to-top. For example, in the sample script shown in Figure 6, T6, T7, T8 are connected traces and together create an island. This island should be declared first and then the devices or leads on top of it need to be declared. Each island can be composed of single or multiple traces. All connected traces in the same island needs to be declared at same hierarchy level. The declaration should start with a ‘+’ character and other connected components need to start with a ‘-‘ character. All components in each connected group should be of the same type (i.e., power traces or signal traces).

Each hierarchy level is separated by a ‘tab’ in the script and we currently support up to 3 levels of hierarchy (2 tabs) in this version (Trace->Device->Pin). Also, in the input script, the coordinates should be given as integer values. However, in the constraint table, the fractional constraint value is allowed up to 3rd decimal point.

All of the width and height information for devices or leads are directly read from the corresponding “.part file” mentioned in the **Definition** section. So, these components do not have

width or height information specified in the layout geometry description script, whereas others (e.g., traces and bond wire pads) have width and height fields.

To have a bond wire, the source pad and destination pad of the bond wire needs to be aligned according to the wire orientation. For example, in the sample script (shown in Figure 6), the gate signal of D1 is connected from B9 to B3 (**Connectivity setup script** section). B9 is on top of the gate pad of D1. So, B9 and B3 should have the same y coordinate as this represents a horizontal bond wire connection. The bond wire pad is considered as a point connection in the algorithms. Since vias are connected across multiple layers, via description needs to be declared in multiple layers with same coordinates.

Description of each line in the '#Layout Information' section:

Line1: Layer ID (from layer stack) <space> routing direction (Z+/Z-)

Line2-to-end: each line has several fields separated by <space>:

For all routing paths (Traces) have 7 fields:

1. '+/-' : Connectivity definition character
2. 'ID' : layout component id (T1: Trace 1, T2: Trace 2,etc.)
3. 'type of component': for traces -> power or signal
4. x coordinate: bottom left corner's x coordinate
5. y coordinate: bottom left corner's y coordinate
6. width: width of the rectangle (along x axis)
7. height: height of the rectangle (along y axis)

For all parts (Devices, Leads, Vias) have 5-6 fields:

1. '+/-' : Connectivity definition character
2. 'ID' : layout component id (D1: Device 1, L1: Lead 1, V1: Via 1, etc.)
3. 'type of component': 'Via' for vias

for devices-> name (should match with definition part) (MOS, Diode, IGBT, cap, res, etc.)

for leads-> name of lead (power_lead or signal_lead or neutral_lead)

4. x coordinate: bottom left corner's x coordinate
5. y coordinate: bottom left corner's y coordinate
6. Rotate angle: R90(90°rotation), R180(180°rotation), R270(270°rotation)

For all bonding wire landing points have 5 fields:

1. '+/-' : Connectivity definition character
2. 'ID' : layout component id (B1: Bond wire point 1, B2: Bond wire point 2, etc.)
3. 'type of component': bond wire pads-> power or signal
4. x coordinate: x coordinate
5. y coordinate: y coordinate

In the test cases in **Sample_Designs** folder, the full-custom layout geometry script has two more fields in each bonding wire landing point declaration line, which are width and length. These two fields are optional in this version, as the bonding wire landing points are treated as points not rectangles.

The coordinates of the elements on the same island (connected group) are correlated to each other. To get the best results, this coordinate correlation needs to be minimized in the initial layout description script. Coordinate correlation reduction would minimize better results. To test if the updated constraints and initial layout description script are valid, please generate a minimum-sized solution first. If there is an error in layout generation, the constraint values may not be feasible. Also, if the minimum-sized solution is not a feasible one, there is probably a correlation issue. Try to break correlations in the input script and run again. This can be a trial and error process where the user needs to play with the input script until a feasible

minimum-sized solution is found.

2. **Connectivity setup script:** This is a text (.txt) file that defines all bonding wire and via connections. An example of the file content for the 2D layout, and 3D layout are shown in Figure 8, and Figure 9, respectively. This file has two parts:

(a) Definition

This section starts with a tag ‘# Definition’. In this section, different types of wire file (*.wire) and via (*.part) locations are described. In each line, there are two segments:

<segment1> <space> <segment2>

<segment1> is the wire/Via name (used to map in the **Table_info** section).

<segment2> is the relative location of the wire file (.wire) or via file (.part).

(b) Table_info

From the initial layout, the user needs to connect each wire/via between its respective landing points/pads. Some of the pin names are mentioned inside the component (*.part) file. For example, a MOS with keyword D1 will have 3 pins: D1_Drain, D1_Source, and D1_Gate. Also, other pins are shown in the layout file (Figure 6), such as B1, B2,..., etc. are used as bonding wire landing pads.

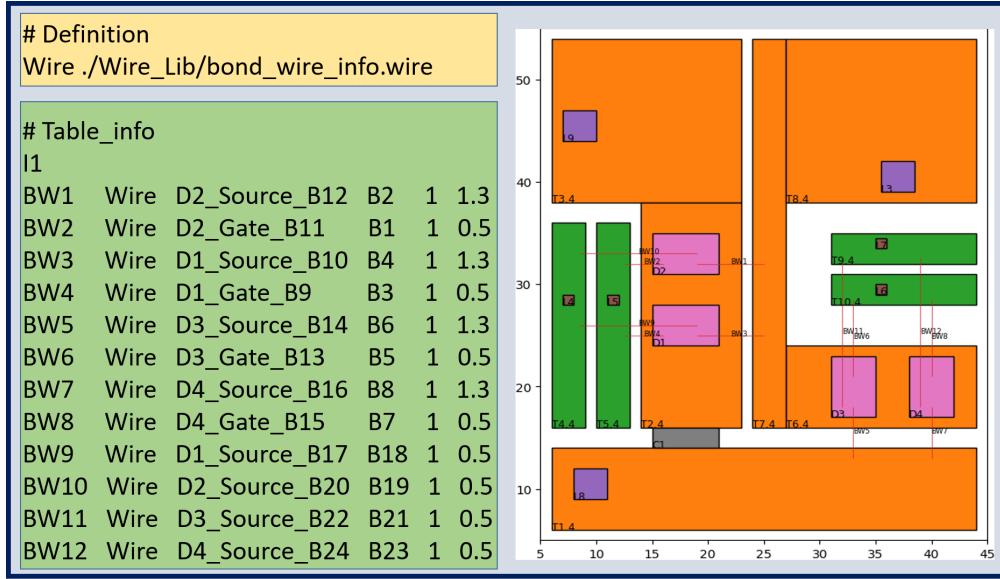


Figure 8: Content in a bonding wire connection description file

Similar to the **Definition** for the components in the layout script, the user can use any keywords for different types of wires. For each row under the **# Table_info** tag, the user can define a bonding wire group (a set of parallel wires), begin with a name (must start with ‘BW’) for the group, wire definition, start pin name, end pin name, number of wires, and spacing (in mm) among multiple wires in the group separated by a space. For the example shown in Figure 8, the first line in the **# Table_info** section defines a bonding wire with name: ‘BW1’, type: ‘Wire’, start pin: ‘D2_Source_B12’, end pin: ‘B2’, number of wires: ‘1’, and spacing: ‘1.3’. Since B12 and B2 are connected, in the layout script (shown in Figure 6) they have same Y-coordinate as this wire is a horizontal wire. Also, B12 is a point on the source pad of the device D2, so the start pin name is ‘D1_Source_B4’. Similarly, for the via connections, user needs to declare 4 fields separated by space: 1. Via connection name (must start with ‘VC’), 2.

Keyword ‘Via’ to match with the part file defined in the **Definition** section, 3. Starting pad name with layer iD in the format <Layout component id><.><Routing layer id>[‘_’: if the routing layer direction is ‘Z-’], 4. Ending pad name with layer id in the same format. Sample example is shown in Figure 9. Since vias are connected across multiple layers, declaration in one layer is enough.

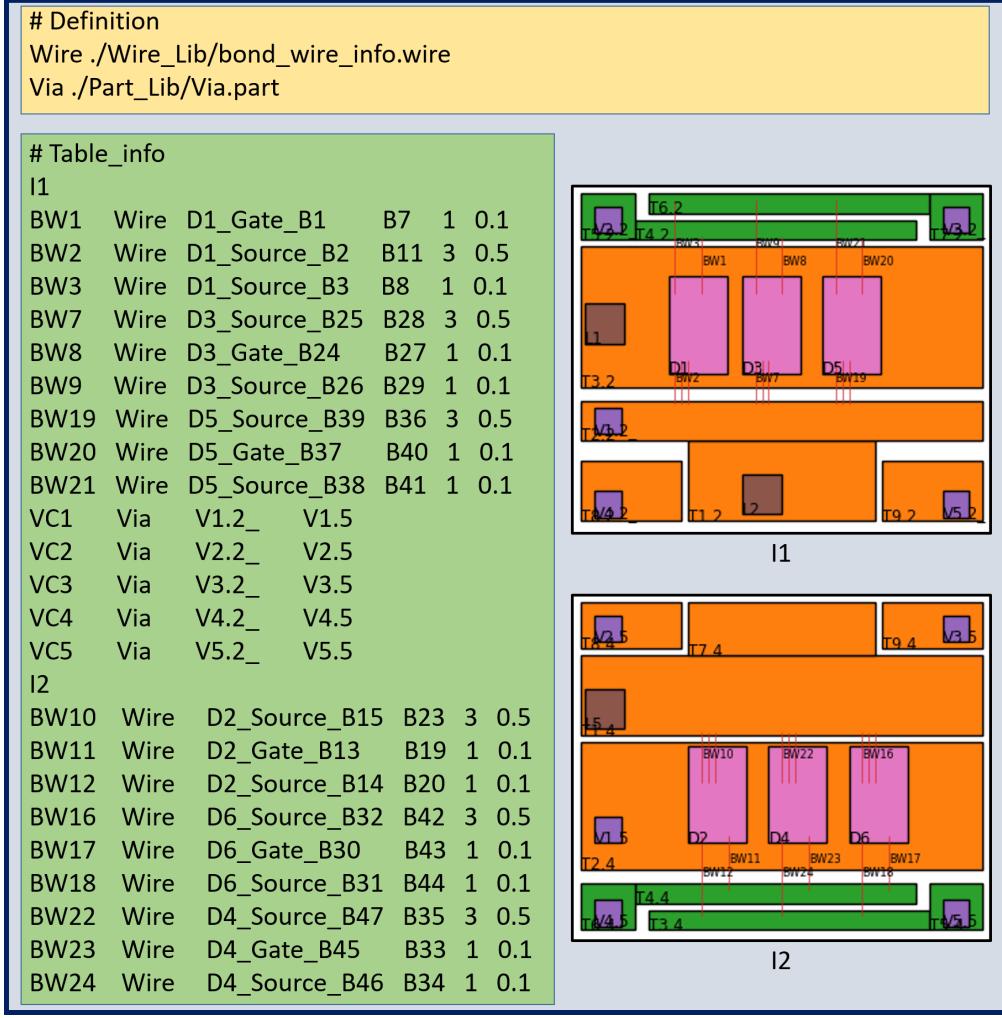


Figure 9: Content in a bonding wire connection description file

(b) Semi-Custom Mode Script: This mode has less flexibility with designing the initial layout. On the other hand, this script preparation requires less effort compared to the full-custom script. In this script, wire bond point locations are automatically calculated based on the corresponding parent component location. No separate connectivity script is required. Everything regarding the layout description is declared in the same script. A sample 2D layout, and 3D layout, the scripts are shown in Figure 10, and Figure 11, respectively. To merge the **Connectivity setup script** in the **Layout geometry description script**, an additional field is added in the **Layout geometry description script** compared to the full-custom mode script. Each section of the script is described below.

(a) Definition

This section is exactly the same as the **Full-Custom Mode Script**. Please see above (**Definition**)

for details. Only difference is the ‘Wire’ definition. Since connectivity setup script is not required and those information have been merged into this script, the ‘Wire’ definition is added to this section along with other component definitions.

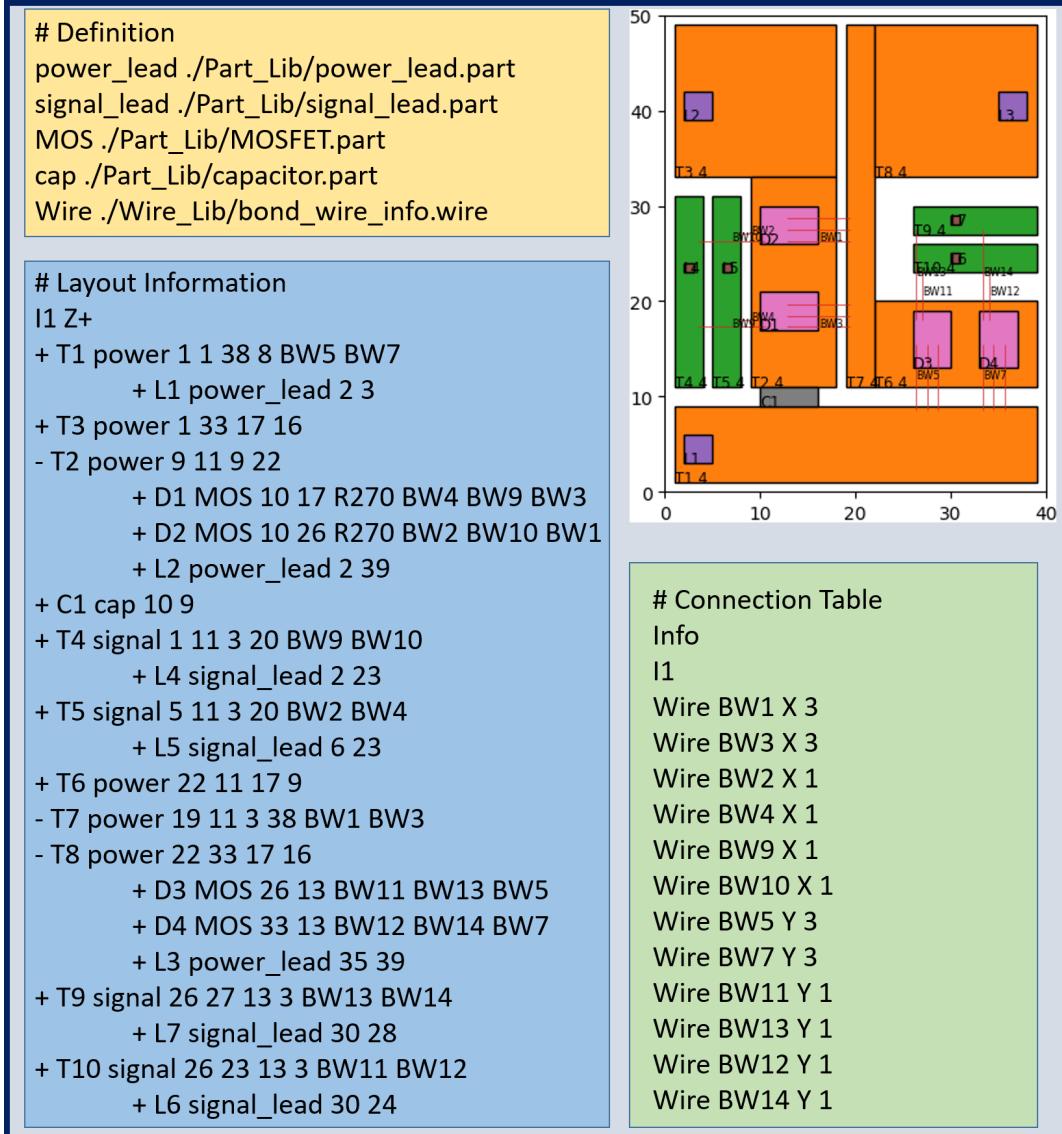


Figure 10: Content in a 2D semi-custom layout geometry description script

(b) Via Connectivity Information

To remove the redundant information, the via connectivity description has been modified. In this section, there are two major fields: 1. Names of the routing layers pair separated by a space, 2. List of the vias separated by space connecting the pair of the layers. A sample example is shown in Figure 11. The via type is not required in this format.

```

# Definition
Via ./Part_Lib/Via1.part
power_lead ./Part_Lib/power_lead1.part
MOS ./Part_Lib/MOSFET1.part
Wire ./Wire_Lib/bond_wire_info.wire

# Via Connectivity Information
I1 I2: V1 V2 V3 V4 V5

# Layout Information
I1 Z-
+ T1 power 15 7 14 6
- T2 power 7 13 30 3 BW2 BW5 BW8
    + L2 power_lead 19 7.5
    + V1 Via 8 13.5
+ T4 signal 11 27 21 2.5 BW1 BW4 BW7
- T5 signal 7 27 4 4.5
    + V2 Via 8 27.5
+ T6 signal 12 30 21 1.5 BW3 BW6 BW9
- T7 signal 33 27 4 4.5
    + V3 Via 34 27.5
+ T3 power 7 17 30 9.5
    + L1 power_lead 7.25 21
    + D1 MOS 13.5 18 BW1 BW3 BW2
    + D3 MOS 19 18 BW4 BW6 BW5
    + D5 MOS 25 18 BW7 BW9 BW8
+ T8 power 7 7 7.5 4.5
    + V4 Via 8 7.25
+ T9 power 29.5 7 7.5 4.5
    + V5 Via 34 7.25

I2 Z+
+ T1 power 7 21.5 30 5 BW11 BW14 BW17
- T7 power 15 26.5 14 5
    + L5 power_lead 7.25 22
+ T4 signal 11 9 21 1.5 BW10 BW13 BW16
- T6 signal 7 7 4 3.5
    + V4 Via 8 7.25
+ T3 signal 12 7 21 1.5 BW12 BW15 BW18
- T5 signal 33 7 4 3.5
    + V5 Via 34 7.25
+ T2 power 7 11.5 30 9.5
    + V1 Via 8 13.5
    + D2 MOS 15 13.5 R180 BW10
    BW12 BW11
    + D4 MOS 21 13.5 R180 BW13
    BW15 BW14
    + D6 MOS 27 13.5 R180 BW16
    BW18 BW17
+ T8 power 7 27 7.5 4.5
    + V2 Via 8 27.5
+ T9 power 29.5 27 7.5 4.5
    + V3 Via 34 27.5

# Connection Table Info
I1
Wire BW1 Y 1
Wire BW2 Y 3
Wire BW3 Y 1
Wire BW4 Y 1
Wire BW5 Y 3
Wire BW6 Y 1
Wire BW7 Y 1
Wire BW8 Y 3
Wire BW9 Y 1

I2
Wire BW10 Y 1
Wire BW11 Y 3
Wire BW12 Y 1
Wire BW13 Y 1
Wire BW14 Y 3
Wire BW15 Y 1
Wire BW16 Y 1
Wire BW17 Y 3
Wire BW18 Y 1

```

Figure 11: Content in a 3D wire-bonded semi-custom layout geometry description script

(c) Layout Information

This section has all the basic information very similar as the **Full-Custom Mode Script**. The key differences are explained here.

1. No wire bond point location is required. If an element of the layout contains a wire bond group, mentioning the group name at the end of geometry description is required. However, if there are multiple wire bond groups land on the same element, the group names need to be inserted sequentially (on a device) or randomly (for other cases except devices. i.e, traces) separated by space. For example, in the layout shown in Figure 10, ‘D1’ MOS has three wire bond groups (BW4, BW9, BW3). These groups need to be inserted in a specific order: Gate wire group, Kelvin source wire group, and Source wire group for the power loop. If any device does not have a Kelvin source wire group, the order needs to be Gate wire group and Source wire group for the power loop. In PowerSynth 2, semi-custom mode script, it is assumed that each MOSFET will have at least two connections (Gate and Source). And vertical devices are considered only (SiC MOS, IGBT, SiC Diode etc.) Since ‘BW4’ is connected between ‘D1’ and ‘T5’, ‘BW4’ is inserted on ‘T5’ trace geometry description line as well. Comparing with the **Full-Custom Mode Script**, it is clear that the bonding wire landing points are removed from the hierarchy script and wire group names have been added at the end of each geometry description line. However, the wire group name declaration order does not matter for the traces and the order needs to be maintained strictly for the devices.
2. Another difference is the via component description. If any via is a through ceramic type, that via needs to be declared on each layer, which is exactly the same as the **Full-Custom Mode Script**. However, if there is any metallic post type connection that connects a device source/gate pad to another routing layer, no location specification of such vias is required. For example, a wire bondless 3D layout geometry script is shown in Figure 12. In this example, both through ceramic via (‘V3’) and metallic post type vias are available. From the layout script, it is clear that the through ceramic via ‘V3’ requires location specification, which is exactly the same as **Full-Custom Mode Script**. However, the rest of the vias do not require any location specification. They are declared in the same way as the wire bond groups. For the devices, the order of declaration needs to be same as wire bond group declaration (Gate, Source) and for the traces, the order does not matter.

Apart from these exceptions, Layout Information description is same as the **Full-Custom Mode Script**.

(d) Connection Table Info

This section is only for the wire bond group description. Connectivity information is embedded in the geometry script. The sample content is shown in Figure 11. The keyword ‘# Connection Table Info’ needs to be there. Then, the layer id is declared. The rest of the following lines describe the wire bond group information until another layer id appears. Each line in the wire bond group description has four fields: 1.The wire name (used to map in the **Definition** section). 2. Wire bond group name that must start with ‘BW’ (i.e., BW1, BW2, etc.). 3. Direction of the wire group (X: horizontal, Y: vertical). 4. Number of wires in each group. The spacing of the wires is calculated from the wire landing pad dimensions and number of the wires. if there is a consecutive series of wire bond groups with same direction and number of wires in the group, rather than describing in multiple lines, the user can merge them using the following order: 1. The wire name, 2. Starting group of the series, 3. Ending group of the series, 4. Direction of the group, 5. Number of wires in each group. For example, if BW1, BW2, BW3 have the same wire name (Wire) , direction (Y), and number of wires (3), then they can be declared by the following line:

Wire BW1 BW3 Y 3

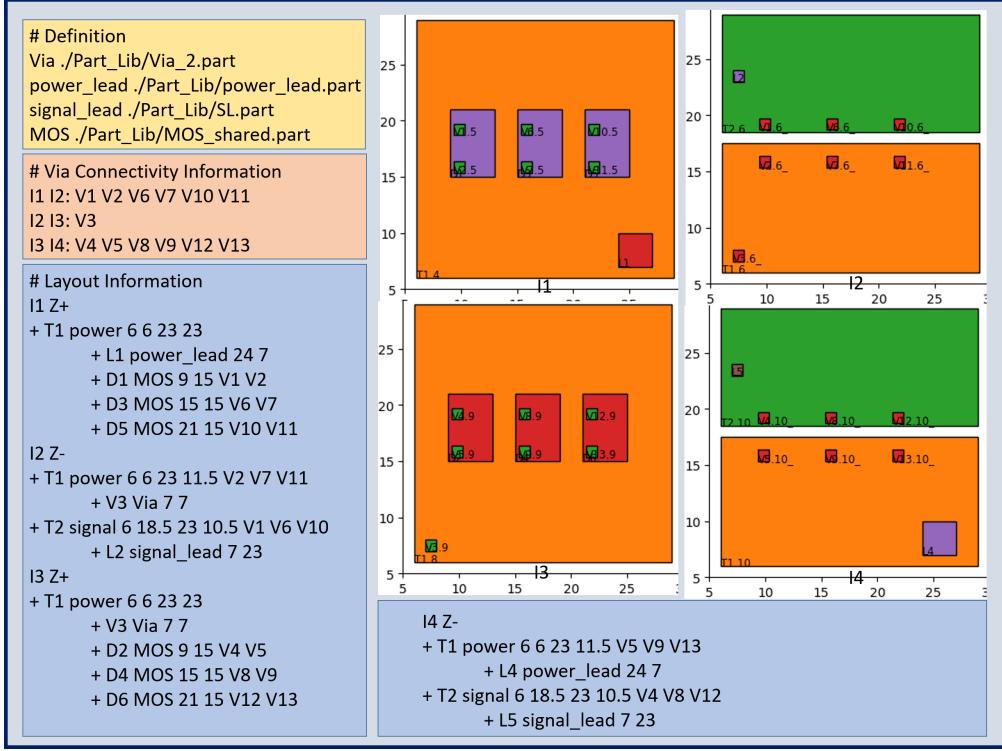


Figure 12: Content in a wire bondless 3D semi-custom layout geometry description script

2.2.3 Constraints

The minimum constraint values are given as input through a CSV file. Generally, for each layout the constraint file is automatically populated with some default values. However, the user can always modify the values according to the manufacturer requirements. Since in this version, two types of constraints (i.e., standard design constraints, and reliability constraints) are considered, the default constraint table generates minimum standard design constraints and the user needs to set a flag (**‘Reliability-awareness’**) to indicate that the reliability constraints are available (high voltage application). The default constraint file content for the sample layout in Fig. 6 is shown in Fig. 13.

In Figure 13, the highlighted (red/green) fields are representing constraint names. The blue colored fields are representing the elements in the layout. All elements are directly related to the layout geometry description script (shown in Figure 6) except the "EMPTY" field. This type represents the etched area on top of a DBC, which means any area where there is no copper. Rest of the fields are dynamically populated based on the elements in the layout.

If the **Reliability-awareness** flag is set, then along with the minimum design constraints shown in Figure 13, additional content is populated in the same constraint file. The additional rows in the csv file for the sample layout in Figure 6 is shown in Figure 14. Here, first few lines are for voltage and current specifications with default value 0. The user needs to modify the values according to the specific requirements. In the voltage and current specification sections, there are four fields to describe the voltage and current loading for each island (connected group of traces) as all waveforms are considered in generic sinusoidal form: $A + B \sin(2\pi ft + \theta)$. Here, A= DC magnitude (V/A), B= AC magnitude (V/A), f= Frequency, θ = Phase angle. So, while providing the values, the waveform needs to be fit in the equation and then the coefficient values need to be provided. If there are multiple traces in an island, one trace will be appeared in the constraint table as all traces on the

same island are connected. For example, in the constraint table shown in Figure 14, only **T2.4** has appeared as **T2.4**, **T3.4**, and **T4.4** are on same island in Figure 6.

Min Dimensions	EMPTY	power_trace	signal_trace	bonding wire pad	power_lead	signal_lead	MOS	cap
MinWidth	1	2	2	0	3	1	4	6
MinLength	1	2	2	0	3	1	6	2
MinHorExtension	1	2	2	0	3	1	4	6
MinVerExtension	1	2	2	0	3	1	6	2
MinHorEnclosure	EMPTY	power_trace	signal_trace	bonding wire pad	power_lead	signal_lead	MOS	cap
EMPTY	1	1	1	1	1	1	1	1
power_trace	1	1	1	1	1	1	1	1
signal_trace	1	1	1	1	1	1	1	1
bonding wire pad	1	1	1	1	1	1	1	1
power_lead	1	1	1	1	1	1	1	1
signal_lead	1	1	1	1	1	1	1	1
MOS	1	1	1	1	1	1	1	1
cap	1	1	1	1	1	1	1	1
MinVerEnclosure	EMPTY	power_trace	signal_trace	bonding wire pad	power_lead	signal_lead	MOS	cap
EMPTY	1	1	1	1	1	1	1	1
power_trace	1	1	1	1	1	1	1	1
signal_trace	1	1	1	1	1	1	1	1
bonding wire pad	1	1	1	1	1	1	1	1
power_lead	1	1	1	1	1	1	1	1
signal_lead	1	1	1	1	1	1	1	1
MOS	1	1	1	1	1	1	1	1
cap	1	1	1	1	1	1	1	1
MinHorSpacing	EMPTY	power_trace	signal_trace	bonding wire pad	power_lead	signal_lead	MOS	cap
EMPTY	1	1	1	1	1	1	1	1
power_trace	1	1	1	1	1	1	1	1
signal_trace	1	1	1	1	1	1	1	1
bonding wire pad	1	1	1	1	1	1	1	1
power_lead	1	1	1	1	1	1	1	1
signal_lead	1	1	1	1	1	1	1	1
MOS	1	1	1	1	1	1	1	1
cap	1	1	1	1	1	1	1	1
MinVerSpacing	EMPTY	power_trace	signal_trace	bonding wire pad	power_lead	signal_lead	MOS	cap
EMPTY	1	1	1	1	1	1	1	1
power_trace	1	1	1	1	1	1	1	1
signal_trace	1	1	1	1	1	1	1	1
bonding wire pad	1	1	1	1	1	1	1	1
power_lead	1	1	1	1	1	1	1	1
signal_lead	1	1	1	1	1	1	1	1
MOS	1	1	1	1	1	1	1	1
cap	1	1	1	1	1	1	1	1

Figure 13: Content in a constraint file with minimum design constraints only

Once the waveforms are defined, the voltage-dependent minimum spacing and current-dependent minimum width values need to be declared in the last two sections (highlighted in red color). For voltage difference vs minimum spacing, user can add rows depending on the expected voltage difference levels in the layout. The voltage difference calculation method can be found in [15]. In the initial constraint table, only one row under ‘Voltage Difference’ and ‘Current Rating’ will appear. For additional constraints, the user needs to add the rows in the CSV file. For example, if the user is expecting 3 additional voltage differences like 5000V, 10000V, 15000 three more rows need to be

Voltage Specification				
Component Name	DC magnitude	AC magnitude	Frequency (Hz)	Phase angle (degree)
T1.4	0	0	0	0
T2.4	0	0	0	0
T6.4	0	0	0	0
Current Specification				
Component Name	DC magnitude	AC magnitude	Frequency (Hz)	Phase angle (degree)
T1.4	0	0	0	0
T2.4	0	0	0	0
T6.4	0	0	0	0
Voltage Difference	Minimum Spacing			
0	2			
Current Rating	Minimum Width			
0	2			

Figure 14: Content in a constraint file associated with the reliability constraints

added in this section with the corresponding minimum spacing values. Similarly, for the current rating vs minimum width constraints, user needs to add rows depending on the requirements. The voltage differences and current ratings in the constraint table need to increase with a constant value. If the calculated voltage difference between two traces is in between two values provided in the constraint table, it will choose the upper bound for safety. For example, if the user has 0 V, 5000 V, 10000 V voltage differences, and the calculated voltage difference is 2500 V, the constraint value that will be applied is for the 5000 V difference case.

The user can open the constraint file and edit the values and save it in the same location. No renaming is required. While assigning the constraint values, care needs to be taken to make sure the constraints are valid.

2.3 PowerSynth 2.0 GUI Introduction

Upon running PowerSynth executable, the following welcome window (shown in Figure 15) will appear. The window has four buttons: (a) **Open Website** button to access PowerSynth webpage,(b) **Open Manual** button to open the manual, (c) **Create a Macro** button to start a new test case flow that will generate a macro script, which can be used later to avoid the tedious GUI flow, and (d) **Run PowerSynth 2** to re-run an existing test case using the macro script generated by the **Create a Macro** flow.



Figure 15: PowerSynth 2.1 welcome window

PowerSynth 2 user can run PowerSynth in two ways: Creating a new Macro, which is the recommended flow to start any new design optimization; Running an existing Macro, which helps the user to avoid tedious GUI flow when the user wants to run the same project with a little modification. Both of these flows are described below.

(a) **Create a Macro:** Upon clicking on this button, the user will be asked to input the initial layout structure information.

- **Data Input:** The fields in the window shown in Figure 16 need to be populated.

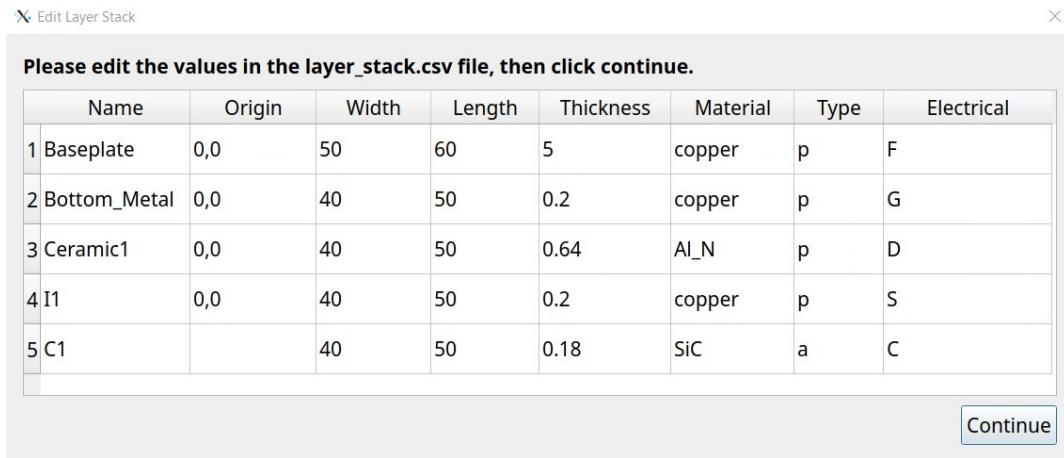
 A screenshot of the 'Initial Structure and Layout' dialog box. It has sections for 'Required' and 'Optional' inputs. Under 'Required', there are fields for 'Layer Stack' and 'Layout Script', each with an 'Open' button. Under 'Optional', there are fields for 'Connectivity Script' and 'Materials' (containing the path '/e3da/tool/e3da/linux/ialrazi/2.0/PowerSynth2.0/pkg'), each with an 'Open' or 'Edit' button. Below these is a 'Reliability Constraints' dropdown set to 'no constraint'. At the bottom right is a 'Create Layout' button.

Figure 16: Data input

The user needs to provide the path to the **Layer stack**, **Layout geometry script** in the corresponding field. Since PowerSynth 2 has both **Full-Custom** and **Semi-Custom** mode of layout geometry script, the **Connectivity script** is set as optional, which means it is required for only **Semi-Custom** mode layout geometry script cases. If the layout geometry script provided by the user is in **Full-Custom** mode, this **Connectivity _script** field should be

left as blank. Each of these script need to be prepared beforehand according to the description in Section 2.2. Finally, the user needs to set up the reliability constraint information. On the drop down list, there are three options: (1) no constraints, (2) average case, (3) worst case. Please note that if the user wants the reliability constraints to be applied, which means if the selection is either average case or worst case, the **constraints.csv** file in the test case folder needs to be populated with the reliability constraints fields as shown in 13 along with the design constraints. If the reliability constraints is set to **no constraints**, only the design constraints need to be populated in the **constraints.csv** file. After choosing one option, the user needs to click the **Create Layout** button to proceed.

- **Edit Layer Stack:** In this stage, the layer stack table will appear and the user can edit the values if they want (shown in Figure 17).



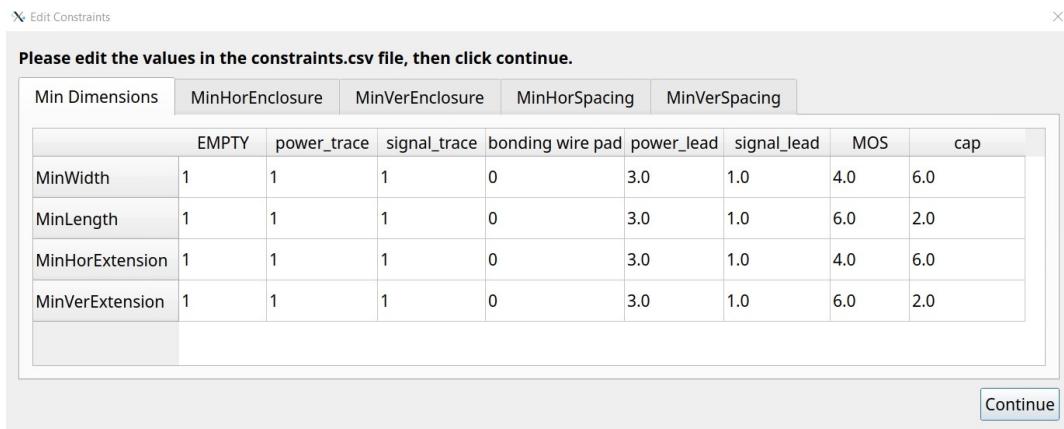
The screenshot shows a window titled "Edit Layer Stack". Inside, a message says "Please edit the values in the layer_stack.csv file, then click continue." Below is a table with columns: Name, Origin, Width, Length, Thickness, Material, Type, and Electrical. The table contains five rows of data:

Name	Origin	Width	Length	Thickness	Material	Type	Electrical
1 Baseplate	0,0	50	60	5	copper	p	F
2 Bottom_Metal	0,0	40	50	0.2	copper	p	G
3 Ceramic1	0,0	40	50	0.64	Al_N	p	D
4 I1	0,0	40	50	0.2	copper	p	S
5 C1		40	50	0.18	SiC	a	C

A "Continue" button is at the bottom right.

Figure 17: Edit Layer Stack

- **Edit Constraints:** Upon clicking on the Continue button, the Edit Constraints window will appear. The user can edit the values if they want (shown in Figure 18).



The screenshot shows a window titled "Edit Constraints". Inside, a message says "Please edit the values in the constraints.csv file, then click continue." Below is a table with columns: Min Dimensions, MinHorEnclosure, MinVerEnclosure, MinHorSpacing, and MinVerSpacing. The table has sub-headers for each row. The table contains four rows of data:

Min Dimensions		MinHorEnclosure	MinVerEnclosure	MinHorSpacing	MinVerSpacing				
		EMPTY	power_trace	signal_trace	bonding wire pad	power_lead	signal_lead	MOS	cap
MinWidth	1	1	1	0		3.0	1.0	4.0	6.0
MinLength	1	1	1	0		3.0	1.0	6.0	2.0
MinHorExtension	1	1	1	0		3.0	1.0	4.0	6.0
MinVerExtension	1	1	1	0		3.0	1.0	6.0	2.0

A "Continue" button is at the bottom right.

Figure 18: Edit Constraints

- **PowerSynth Run Options:** The next step would be choosing an option for using Power-

Synth from three following choices (shown in Figure 19). These options are referred to as **Option** in the macro script.

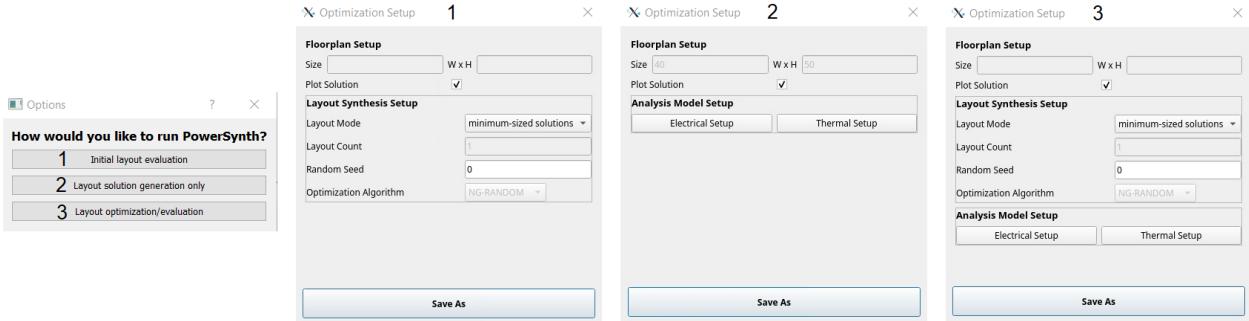


Figure 19: PowerSynth 2 run options

1. Layout solution generation only: Since PowerSynth 2 layout engine is a generic, scalable, and efficient one to generate the layout solutions rapidly, user can use this flow. However, in this mode, no layout performance evaluation will be performed. This mode of operation is referred to as **Option 0** in the macro script. As shown in the window labeled as 1 in the Figure 19, the user can generate three types of layout solutions (described in Table 1). The user needs to provide the randomization seed so that the solution generation can be repetitive. For the minimum-sized solutions (**layout mode 0**), there will be a single solution, whereas for other two cases, the **Number of layouts and generations** needs to be provided. For the Fixed-sized solutions case (**layout mode 2**), the **Floor Plan** size needs to be provided as well. Please note that the floorplan size needs to be greater than the minimum floorplan size. For solution generation, the default algorithm is the non-guided randomization (NG-RANDOM).
2. Initial layout evaluation: This flow will evaluate the initial layout described in the layout geometry. Please note that no design rule checking will be performed on this layout. This option is called **Option 1** in the macro script. Upon clicking on this button the window marked as 2 in the Figure 19 will appear. From this window, user needs to set up electrical and thermal modeling parameters by clicking on the corresponding buttons. Once those are set up, save the macro script using **Save As** button. After that using **Run PowerSynth 2** button will start the evaluation and take to the **Solution Browser**.
3. Layout optimization/evaluation: This step is the most useful one that is referred to as **Option 2** in the macro script. In this flow, the user can generate DRC-clean solutions and evaluate the electro-thermal performance values. The electrical and thermal setup steps are described in the next step. In this mode, if user is using fixed-sized solutions or variable-sized solutions, then the user can choose either **NG-RANDOM** or genetic algorithm **NSGAII** particle swarm (**MOPSO**) for optimization. If the choice is **NG-RANDOM**, the number of layouts need to be provided in the **Layout count** field. Otherwise, in addition to number of layouts, the number of generations must be determined in the **Number of Generations** for **NSGAII** and **MOPSO**.

- **Electrical Setup:** If the user wants to evaluate/optimize the layout by choosing either **Option 1** or **Option 2**, the electrical setup window (shown in Figure 20) needs to be opened and necessary parameters need to be provided.

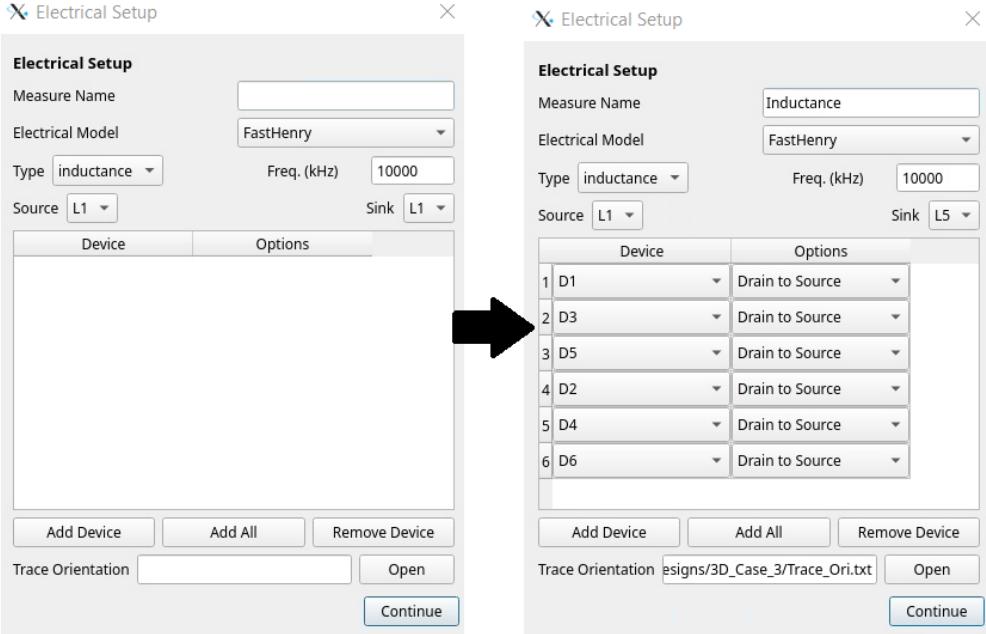


Figure 20: Electrical setup window: Before (Initial) and after (Final) setting up parameters

In this window, the user can choose the appropriate electrical model for evaluation from two options: FastHenry (for 3D layouts) and PEEC (for 2D) layouts. Then, the user needs to provide a name of the performance measure in the **Measure Name** field. User can choose either inductance or resistance for parasitic extraction. Then, the devices in the layout need to be setup for completing the desired loop. If it is the power loop, all devices' drain and source need to be connected. To do so, the user needs to click on **Add All** button. That will add all devices in the design in the table as shown in the Figure 20. The user can define the partial loop by choosing selective devices and making sure the loop is complete by choosing appropriate connection corresponding to the device. **Add Device** button will add devices one by one and **Remove Device** button can be used to remove any device from the table. Then, the user needs to choose 'Source' and 'Sink' for the loop. All leads will be appeared in the drop down list. To make sure the power/signal loop connectivity is established, for each device, appropriate option need to be selected. For example, for the power loop connection, all the vertical devices' drain and source need to be connected. The parasitic extraction frequency can be set up in the **Frequency** field. Please note that the value is assumed to be in the kHz. So, the entered value will be multiplied by 1000 for actual frequency in Hz. Finally, the trace orientation file needs to be inserted. The content in a sample trace orientation file for the layout shown in Figure 6 is shown below.

H: T1.4,T3.4,T6.4,T8.4,T9.4,T10.4
V: T2.4,T4.4,T5.4,T7.4

Here, H stands for horizontal and V stands for Vertical. From the layout script, the user should know which traces are horizontally (x-axis dimension \geq y-axis dimension) routed and which are vertically (y axis dimension \geq x-axis dimension). The trace name has two parts: '*<Layout component id>.<Interconnect layer id>*'. Layout component id should come from the layout script and the interconnect layer id should come from the layer stack. After populating all the fields the window should look like the one shown in Figure 20 (right). Then, clicking the **Continue** button will take back to the optimization setup window.

- **Thermal Setup:** For thermal evaluation setup, the parameters need to feed through this window (shown in Figure 21).

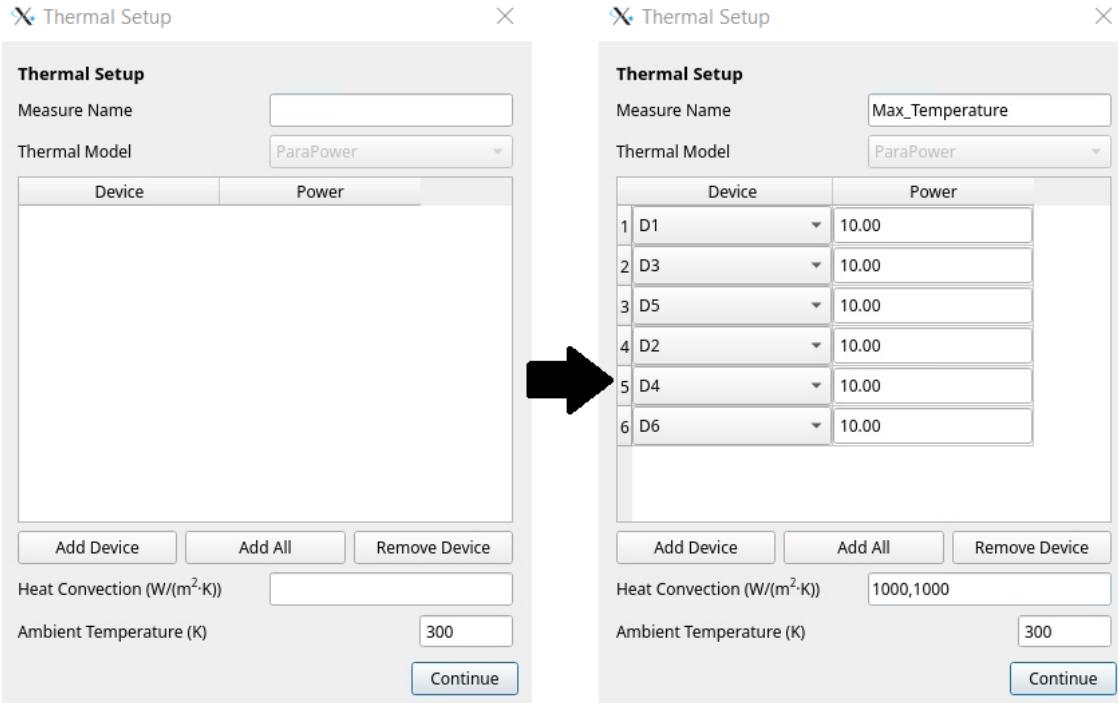


Figure 21: Thermal setup window: Before (Initial) and after (Final) setting up parameters

To facilitate thermal evaluation for 2D/3D layouts, the ‘ParaPower’ model is the default one. The user needs to input a performance name for the measurement. Then, similar to the electrical setup, the user needs to populate the table with the static heat dissipation value for each device. In the **Heat Convection** field, if the module has a single-sided cooling, only a single value of effective heat transfer coefficient value in $W/m^2 - K$ needs to be provided. If the layout has a double-sided cooling, both faces’ heat transfer coefficient value need to be inserted separated by a comma (as shown in Figure 21 (right)). Finally, the user needs to setup the ambient temperature in Kelvin. Upon filling all the parameters, the window will look like Figure 21 (right). Then, clicking the **Continue** button will take back to the optimization setup window.

- **Solution Browser:** Once the electrical and thermal settings are loaded, save the macro script by clicking on **Save As** button. After that, upon running the PowerSynth by clicking on **Run PowerSynth 2** button, and **Run** button in the Run Macro window, the interactive solution browser window will pop up with the solution space. A sample solution space with a single solution is shown in Figure 22. On the window, there is a layout viewer on the left side. Each dot in the solution space is clickable and upon clicking on each dot, the corresponding layout will appear in the viewer. If the design is a 3D layout with multiple layers, each layer will have a tab, and to see the via alignment, there is a ‘All Layers’ tab. If the user is running PowerSynth in evaluation/optimization mode, the performance values, and floorplan size of the selected layout will be appeared in the bottom. Finally, the user can either export the selected solution by clicking on **Export Selected** button or export all solutions by clicking

on **Export All**. Upon exporting the solution/solutions, the solution space, individual layout information, and parasitic netlist will be saved in the ‘Solutions’ directory of the design case folder.

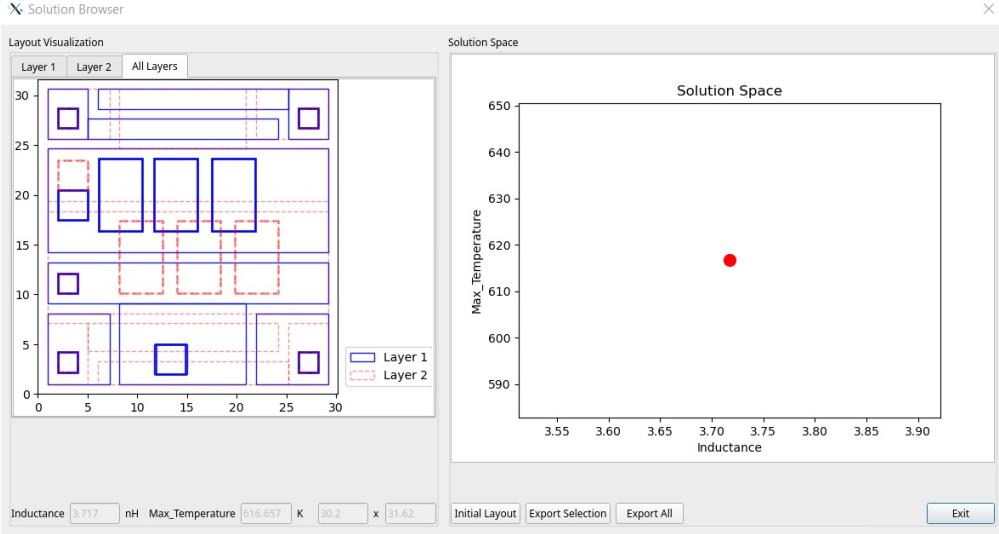


Figure 22: Minimum-sized solution space window

(b) Run PowerSynth: If the user has a macro script ready from the **Create a Macro** flow, the user may use this option from the ‘Welcome’ window. Upon clicking on this button, the window shown in Figure 23 and asks for the **macro script** file from the user. The **macro script** content is discussed below.

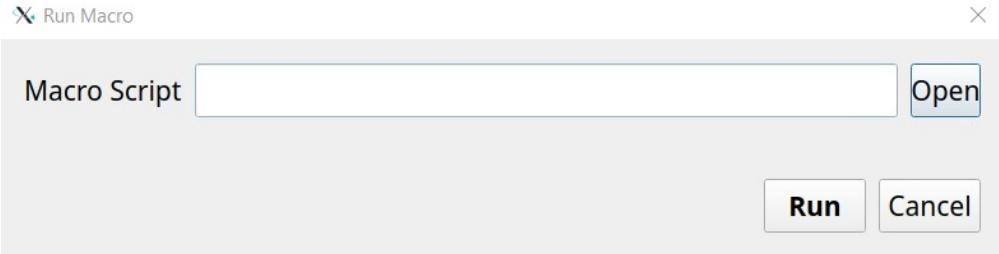


Figure 23: Run a project window

1. Input Scripts
2. Layout Generation and Optimization Setup

Description about each section is as follows: **Input Scripts** In this section, eight files/directories locations are provided. Detailed description for each of them are provided below:

1. **Layout_script:** In this field, the location of the **Layout geometry description script** needs to be provided.
2. **Connectivity_script:** Here, the location of the **Connectivity setup script** needs to be provided for the **Full-Custom Mode Script**. Since the **Semi-Custom Mode Script** does not require the separate **Connectivity setup script**, this field should not be in the macro script for those cases.

3. **Layer_stack:** Location of the **Layer stack** file is provided here.
4. **Parasitic_model:** **default** value should be good here as this is required for **Response surface** electrical model, which is not supported in this version.
5. **Fig_dir:** Needs a path of a directory to save the figures.
6. **Solution_dir:** Needs a path of a directory to save the solution database file. This database file contains layout information which is used to plot the figures. In this folder, all performance values of the solutions and corresponding Pareto-front solution set are reported as a .csv format. Besides, each layout solution components and corresponding coordinates, dimensions of all solutions and Pareto solutions are dumped in individual CSV file in this folder.
7. **Constraint_file:** The constraint file is a CSV file, where all constraint values are stored. Depending on the mode of the flag '**New**' in the macro file, a constraint file will be created or loaded to generate the layout solutions. In this field, the user needs to provide the location of an empty CSV file (for the first time for each layout) and make sure the '**New**' flag is set to 1. This flag value allows user to modify the default constraint values populated by PowerSynth. Once the values are modified, the '**New**' flag needs to be set to 0, which reloads the constraint values in the specified file and does not require to edit the values again.
8. **Model_char:** The path of the folder named 'Characterization' should be entered here. In this folder the device setup connection is stored as a json file, which is used by the electrical model. For the first time run, nothing needs to be in the folder. It will be automatically populated after the first run.
9. **Trace_Ori:** This field is the 'Trace Orientation' description file. A text (.txt) file location with trace orientations needs to be provided. This file is required for the electrical model to evaluate electrical performance. Two orientations are possible for each trace: Horizontal and Vertical. This represents the preferred current flow direction for the trace.

Layout Generation and Optimization Setup Layout Generation

In this section, PowerSynth layout solution generation and performance evaluation setup are defined. Each field for this section is described below:

1. **Reliability-awareness:** If high-voltage and current dependent constraints (reliability constraints) are available and the user wants to apply those, this flag should be set to 2. Setting up this flag as 0 indicates no reliability constraints are applied. If this flag is set to 1, worst-case conditions (theoretical, not always practical) are considered. To have reasonable, practical, and reliability-aware layout generation, this flag needs to be set to 2. If reliability-awareness is set to 1, or 2, in the constraint table, voltage and current specifications are populated for modification. Details are described in Section 2.2.1.
2. **New:** If '**New**' flag is set to 1, user will be asked to edit the constraint table in the given constraint file location while running the test case. Once the edition is done, user needs to enter "1" in the terminal to continue. For each new test case, this flag has to be set to 1 for the first run. Then, for other iterations, the flag needs to be set to 0, otherwise the modified constraint values will be overwritten by the default values. If the flag is set to 0, the constraint values already saved in the given constraint file are used to generate solutions.
3. **Plot_Solution:** If this flag is set to 1, all solution layout images will be saved in the **Fig_dir**.

4. **Option:** To choose from the three options mentioned earlier: 1. Layout solution generation only, 2. Initial layout evaluation, 3. Layout optimization/evaluation, this flag is provided. Based on your choice provide 0, 1, 2 respectively for the options given above. For example: to run layout optimization, you will provide 2 in the option field.
5. **Layout_Mode:** This field is related to layout generation. In this version, three modes of layout generation are allowed: 1. Mode-0: minimum-sized solution, 2. Mode-1: variable-sized solutions, 3. Mode-2: fixed-sized solutions. This option is effective if the user has chosen 0 or 2 as **Option** field value. For Mode-0, a single solution is generated. For other modes user can choose number of solutions. Since in Mode-0, user can generate a single solution, if '**Option**' is set to 2, that single solution will be evaluated and this will not generate any Pareto-front as the solution space length is 1. It is recommended that for each new test case, user generates minimum-sized solution (Mode=0) to make sure that the initial layout and constraint values are correct. However, based on user requirement, **Layout_Mode** value can be 0, 1, or 2. For example: if user chooses fixed size layouts to be generated, **Layout_Mode** should be set to 2. There are some other parameters required depending on the **Layout_Mode**: Seed, Floor_plan, Num_of_layouts and Num_generations.
6. **Seed:** Randomization seed. Effective for **Option**=0, and 2 and **Layout_Mode**= 1, 2. Because for the rest of the mode/option there is a single solution.
7. **Floor_plan:** Size of layout (Width, Height). This input is effective for **Option**=0, and 2 and **Layout_Mode**= 2. Because the Mode-2 generates the fixed floorplan size solutions. Please make sure that the input floorplan size is larger or equal to the minimum floorplan size.
8. **Num_of_layouts:** Desired number of solutions. Need to give an integer. Required for **Option**=0, and 2 and **Layout_Mode**= 1, 2.
9. **Optimization_Algorithm :** It can be either **NG-RANDOM**, **NSGAII**, or **MOPSO**. If the **NSGAII** or **MOPSO** is chosen, please note that **Num_of_layouts** field and **Num_generations** field must be determined for the optimizer.

Performance Evaluation Setup

In this version, two types of performance evaluation are allowed: Electrical and Thermal. This allows electro-thermal optimization only. Also, only two objectives are allowed at a time. In this section, electrical and thermal evaluations are set up. The following information are required for the electrical and thermal APIs:

a) Electrical Setup

This section should start with keyword "**Electrical_Setup:**" and end with keyword "**End_Electrical_Setup.**" In between these two lines following information are required:

1. **Measure_Name:** Put an arbitrary name for electrical measurement.
2. **Model_Type:** Use either of the two options: PowerSynthPEEC or FastHenry.
3. **Measure_Type:** Two options: 1. Inductance, 2. Resistance. Based on the user's choice please provide 0, 1 respectively.
4. **Device_Connection:** The device connection setup is required to complete the loop before evaluating the loop inductance. So, this part defines connections among the pins of each device. For example: MOS has 3 pins: Drain, Source, and Gate. So, there are three options for each MOS: Column1: Drain-to-Source, Column2: Drain-to-Gate, Column3: Gate-to-Source.

Based on your requirement for the desired loop evaluation, provide Device ID and Sequence of 0 or 1 corresponding to each column. For example, in the layout script if you have two MOS declared as

D1 and D2 and you want to short the drain and source of each device, your input will be:

Device_Connection:

D1 1,0,0

D2 1,0,0

End_Device_Connection.

4. Loop Source and Sink: To evaluate a loop inductance or resistance, you need to provide a source and sink. Sources and sinks should be from the leads (L1, L2, ..., etc.)

For example, to measure loop inductance from Lead 1 to Lead 4, the input will be:

Source: L1

Sink: L4

5. Frequency: You need to provide a switching frequency for parasitic extraction. The unit is in kHz.

b) Thermal Setup

In this version, only static maximum temperature can be evaluated. This section should start with keyword **Thermal_Setup:** and end with keyword **End_Thermal_Setup..** In between these two lines, the following information are required:

1. Model_Select: 2 is the value, which needs to be input here to select the **ParaPower** model.
2. Measure_Name: Put an arbitrary name for thermal measurement.
3. Selected_Devices: Enter the list of device names for which user wants to measure their respective temperatures.
3. Device_Power: Enter list of power/heat dissipation (W) for each device in the Selected_Devices.
4. Heat_Convection: Enter a value for the heat transfer coefficient that needs to be applied to the baseplate backside($W/m^2 - K$).
5. Ambient_Temperature: Enter a value of ambient temperature (K).

To summarize the above-mentioned contents, a sample macro script template is shown in Table 3.

Table 3: List of fields in a macro script

# Input Scripts: Layout_script: Connectivity_script: Layer_stack: Parasitic_model: Fig_dir: Solution_dir: Constraint_file: Model_char: Trace_Ori:	# Performance Setup Electrical_Setup: Measure_Name: Model_Type: Measure_Type: Device_Connection: D1 1,0,0 .End_Device_Connection. Source: Sink: Frequency: End_Electrical_Setup.
# Layout Generation Set Up Reliability-awareness: New: Plot_Solution: Option: Layout_Mode: Floor_plan: Num_of_layouts: Seed: Optimization_Algorithm: Num_generations:	Thermal_Setup: Measure_Name: Selected_Devices: Device_Power: Heat_Convection: Ambient_Temperature: End_Thermal_Setup.

2.4 Walk-Through Three Examples

Since PowerSynth 2 can optimize both 2D and 3D layouts, in this section one of each kind is demonstrated step-by-step. In addition, in this version converter layout design which is in the experimental phase is considered as a new example case design.

2.4.1 2D MCPM Layout Optimization

A sample 2D half-bridge power module layout (shown in Figure 24) is chosen to demonstrate the features of PowerSynth v2.1. To optimize the power loop inductance from DC+ (L2)-to-DC- (L1) and the maximum temperature of the layout, the layout script needs to be generated. Also, to represent the layout we need to have the following parts:

1. MOSFET, 2. Power Lead, 3. Signal Lead, 4. Capacitor. The steps are described as follows:

1. Prepare all part files and save in **Part_Lib** folder. So, this folder should have:
 1. MOSFET.part (Content is shown in Listing 1)
 2. Power_Lead.part (Content is shown in Listing 2)
 3. Signal_Lead.part (Content is shown in Listing 3)
 4. capacitor.part (Content is shown in Listing 4)

Listing 1: MOSFET.part file content

```
Name CPM2-1200-0025B
Type component
Link https://www.wolfspeed.com/downloads/dl/file/id/152/
product/2/cpm2\_1200\_0025b.pdf
Footprint 4 6
Thickness 0.18
Material SiC
Pins Drain Source Gate
  Drain 0 0 4 6 B
  Source 0.335 0.35 3.5 4.5 T
  Gate 1.03 5 0.5 0.5 T
Parasitics
  Drain Source R:25e-3
  Drain Gate R:1.1
  Gate Source R:1.1
```

Listing 2: power_lead.part file content

```
Name PL1
Type connector
Footprint 3 3
Thickness 2
Material copper
```

Listing 3: signal_lead.part file content

```
Name SL1
Type connector
Footprint 1 1
Thickness 2
Material copper
```

Listing 4: capacitor.part file content

```
Name CKG57NX7T2J105M500JH
Type component
Footprint 6 2
Thickness 5
Material Al\_N
```

2. Prepare the wire definition file and save in **Wire_Lib** folder. So, this folder should contain the ‘bond_wire_info.wire’ file (content is shown in Listing 5).

Listing 5: bond_wire_info.wire file content

```
JEDEC-4 points
Resistivity 0.000001
Radius 0.127
```

3. Create layout script. To represent the layout shown in Figure 24, the initial **layout_geometry_script** is shown in Listing 7.

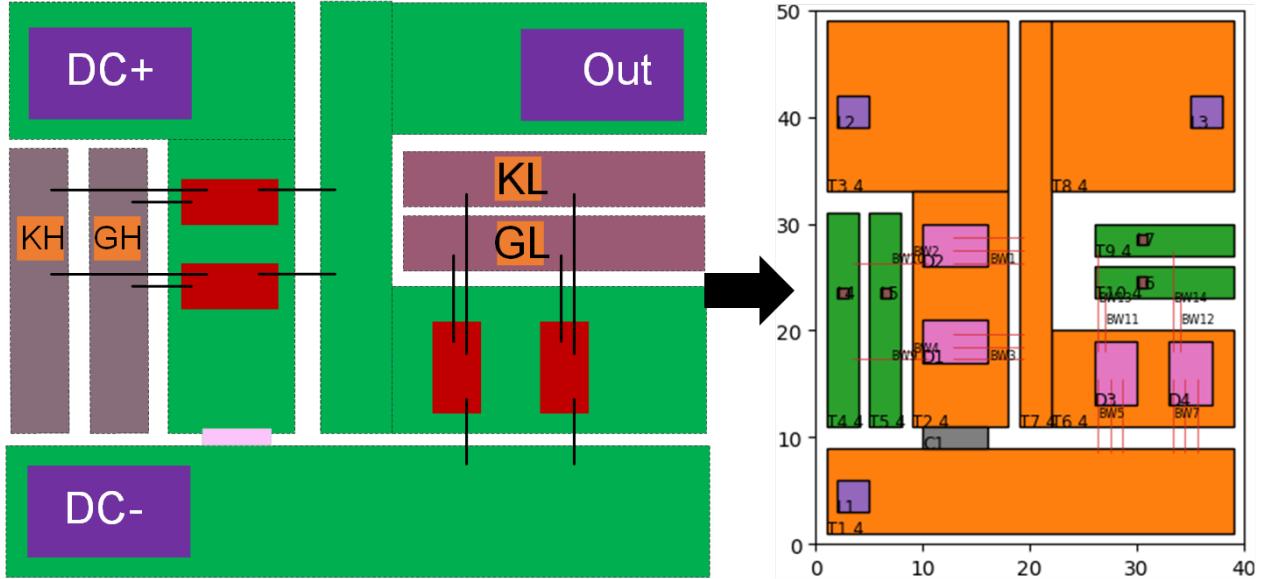


Figure 24: (a) A sample 2D half-bridge power module layout, (b) script-generated layout

4. Prepare layer stack file and save as .csv format. The file content for this design is shown in Table 4.

Table 4: Layer_stack file content

ID	Name	Origin	Width	Length	Thickness	Material	Type	Electrical
1	Baseplate	0,0	50	60	5	copper	p	F
2	Bottom_Metal	0,0	40	50	0.2	copper	p	G
3	Ceramic1	0,0	40	50	0.64	Al_N	p	D
4	I1	0,0	40	50	0.2	copper	p	S
5	C1		40	50	0.18	SiC	a	C

5. Create a blank csv file for constraints. For this case, the file name is "constraint.csv".
6. Prepare the trace orientation file. The content for this design is shown in Listing 14

Listing 6: Trace_Ori file content

```
H:T1.5,T3.5,T6.5,T8.5,T9.5,T10.5
V:T2.5,T5.5,T7.5,T4.5
```

Listing 7: Layout_script for the layout shown in Fig. 24

```
# Definition
power_lead ./Part_Lib/power_lead.part
signal_lead ./Part_Lib/signal_lead.part
MOS ./Part_Lib/MOSFET.part
cap ./Part_Lib/capacitor.part
Wire ./Wire_Lib/bond_wire_info.wire
# Layout Information
I1 Z+
+ T1 power 1 1 38 8 BW5 BW7
    + L1 power_lead 2 3
+ T3 power 1 33 17 16
- T2 power 9 11 9 22
    + D1 MOS 10 17 R270 BW4 BW9 BW3
    + D2 MOS 10 26 R270 BW2 BW10 BW1
    + L2 power_lead 2 39
+ C1 cap 10 9
+ T4 signal 1 11 3 20 BW9 BW10
    + L4 signal_lead 2 23
+ T5 signal 5 11 3 20 BW2 BW4
    + L5 signal_lead 6 23
+ T6 power 22 11 17 9
- T7 power 19 11 3 38 BW1 BW3
- T8 power 22 33 17 16
    + D3 MOS 26 13 BW11 BW13 BW5
    + D4 MOS 33 13 BW12 BW14 BW7
    + L3 power_lead 35 39
+ T9 signal 26 27 13 3 BW13 BW14
    + L7 signal_lead 30 28
+ T10 signal 26 23 13 3 BW11 BW12
    + L6 signal_lead 30 24
# Connection Table Info
I1
Wire BW1 X 3
Wire BW3 X 3
Wire BW2 X 1
Wire BW4 X 1
Wire BW9 X 1
Wire BW10 X 1
Wire BW5 Y 3
Wire BW7 Y 3
Wire BW11 Y 1
Wire BW13 Y 1
Wire BW12 Y 1
Wire BW14 Y 1
```

7. Create a blank csv file for constraints. For this case, the file name is "constraint.csv".
8. Prepare the trace orientation file. The content for this design is shown in Listing 8

Listing 8: Trace _ Ori file content

```
H : T1 .4 , T2 .4 , T4 .4 , T6 .4 , T5 .4 , T10 .4
V : T3 .4
```

9. Now, run PowerSynth executable by running the **PowerSynth2** command from ubuntu 22.04 machine. For Windows machine double click the **PowerSynth2-GUI**. Then, follow the GUI instructions and provide necessary files according to the figures shown below.
10. **Create a Macro Flow:** Start with the **Create a Macro** button and we will be using the default materials. Then, populate the **layer_stack.csv** file location, **layout_script.txt** file location. Since the layout geometry script is in semi-custom mode, the **Connectivity_script** field should be left blank. Since in this run, we will not be using any reliability constraints settings, **None** is selected. These steps are summarized in Figure 25.

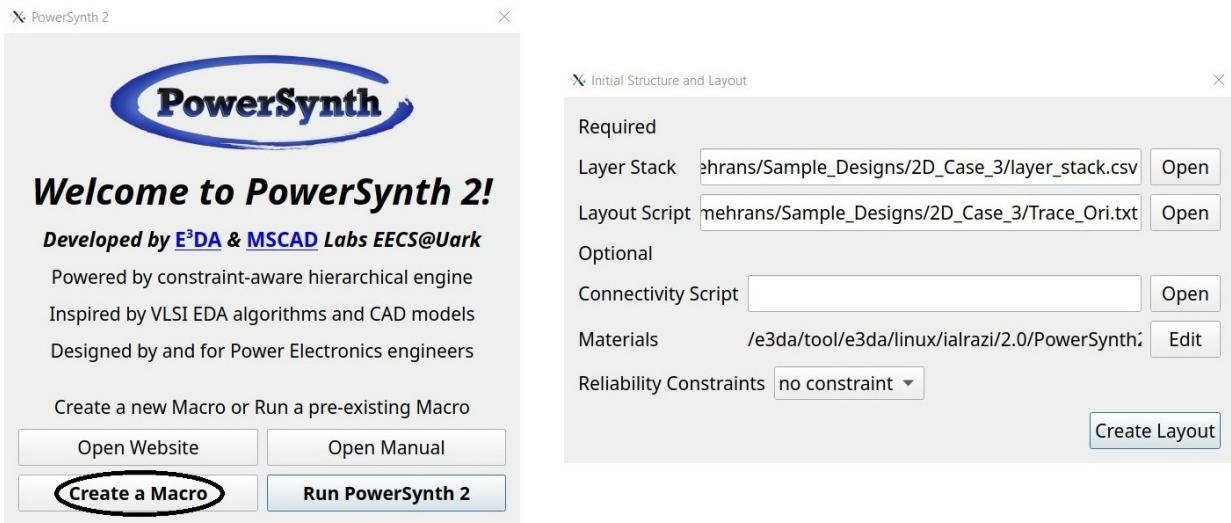
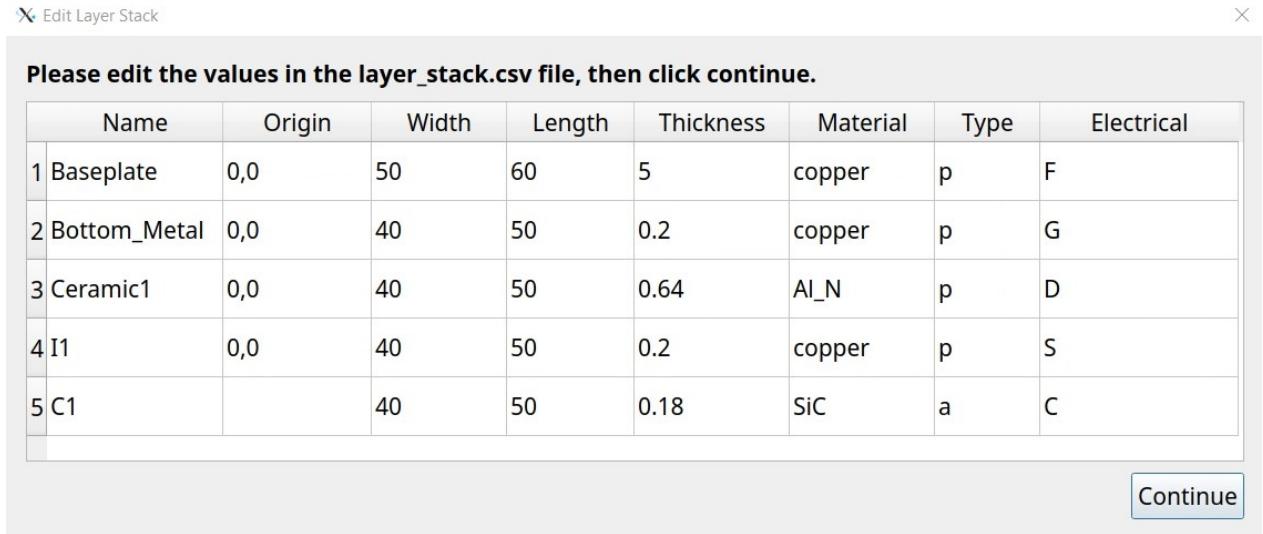


Figure 25: Start PowerSynth and loading module structure

11. Upon clicking on the **Create Layout** button, the **Edit Layer Stack** window will appear. Since we have already created the layer stack file, we will hit **Continue** to proceed (as shown in Figure 26).
12. In this stage, the constraint table will appear and you can edit the default values if you want. In different tabs, different type of constraint values are shown (shown in Figure 27). We will hit **Continue** as we don't want to change any value.
13. Now, we will select the option to run PowerSynth optimization. Since we want to optimize the layout, we will choose the **Layout optimization/evaluation** button to proceed. Before start optimization for a fixed floorplan size, we will evaluate the minimum floorplan size solution.

So, we will input the seed to control randomization. Then, we will click the **Open Electrical Setup** to set up electrical performance evaluation parameters. These steps are summarized in Figure 28.

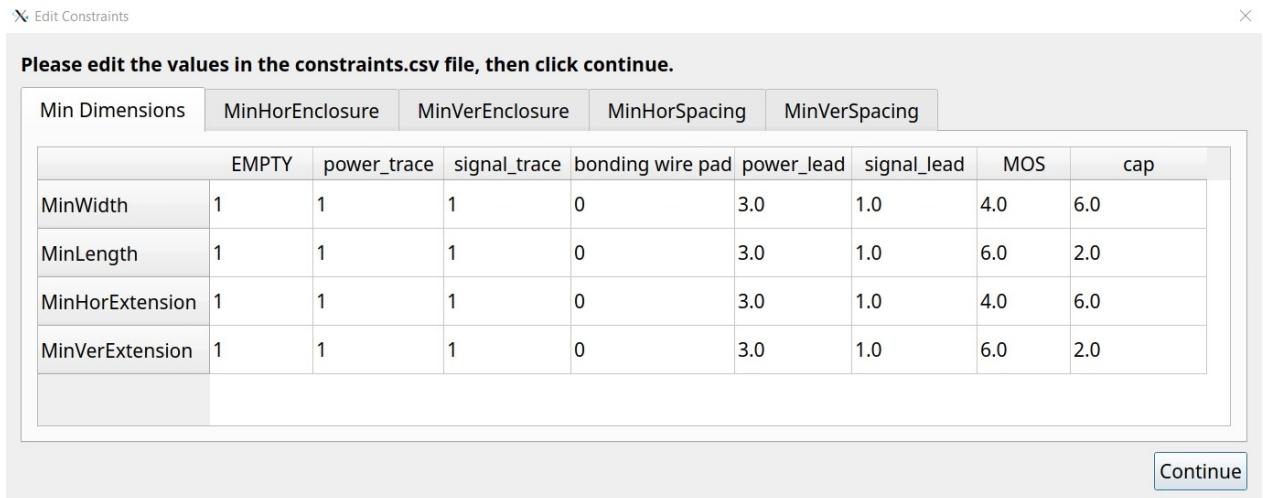


The dialog box has a header 'Edit Layer Stack' and a close button 'X'. It contains the following text: 'Please edit the values in the layer_stack.csv file, then click continue.' Below is a table:

	Name	Origin	Width	Length	Thickness	Material	Type	Electrical
1	Baseplate	0,0	50	60	5	copper	p	F
2	Bottom_Metal	0,0	40	50	0.2	copper	p	G
3	Ceramic1	0,0	40	50	0.64	Al_N	p	D
4	I1	0,0	40	50	0.2	copper	p	S
5	C1		40	50	0.18	SiC	a	C

At the bottom right is a 'Continue' button.

Figure 26: Loaded layer stack



The dialog box has a header 'Edit Constraints' and a close button 'X'. It contains the following text: 'Please edit the values in the constraints.csv file, then click continue.' Below is a table:

	Min Dimensions	MinHorEnclosure	MinVerEnclosure	MinHorSpacing	MinVerSpacing			
	EMPTY	power_trace	signal_trace	bonding wire pad	power_lead	signal_lead	MOS	cap
MinWidth	1	1	1	0	3.0	1.0	4.0	6.0
MinLength	1	1	1	0	3.0	1.0	6.0	2.0
MinHorExtension	1	1	1	0	3.0	1.0	4.0	6.0
MinVerExtension	1	1	1	0	3.0	1.0	6.0	2.0

At the bottom right is a 'Continue' button.

Figure 27: Loaded default constraint table

14. Since it's a 2D layout, we will select the **PEEC** model for parasitic evaluation, and put the measurement name as **Inductance** as we will evaluate **inductance** of the power loop of the module. To set up the power loop we will click on either **Add Device** button four times as there are four devices in the module or **Add All** button once. Then, for each device, we need to select **Drain to Source** to complete the power loop. Then, we have to choose the source and sink of the loop. In this case, DC+ is **L2** and DC- is **L1**. The extraction frequency in kHz needs to be provided. Then, the **Trace_Ori** file location needs to be added. Since we do not have any **parasitic_model** file, we will put **default** there. After entering all of these information **Continue** button will take back to the optimization setup window. From

there, the **Open Thermal Setup** button needs to be clicked to setup thermal performance evaluation parameters.

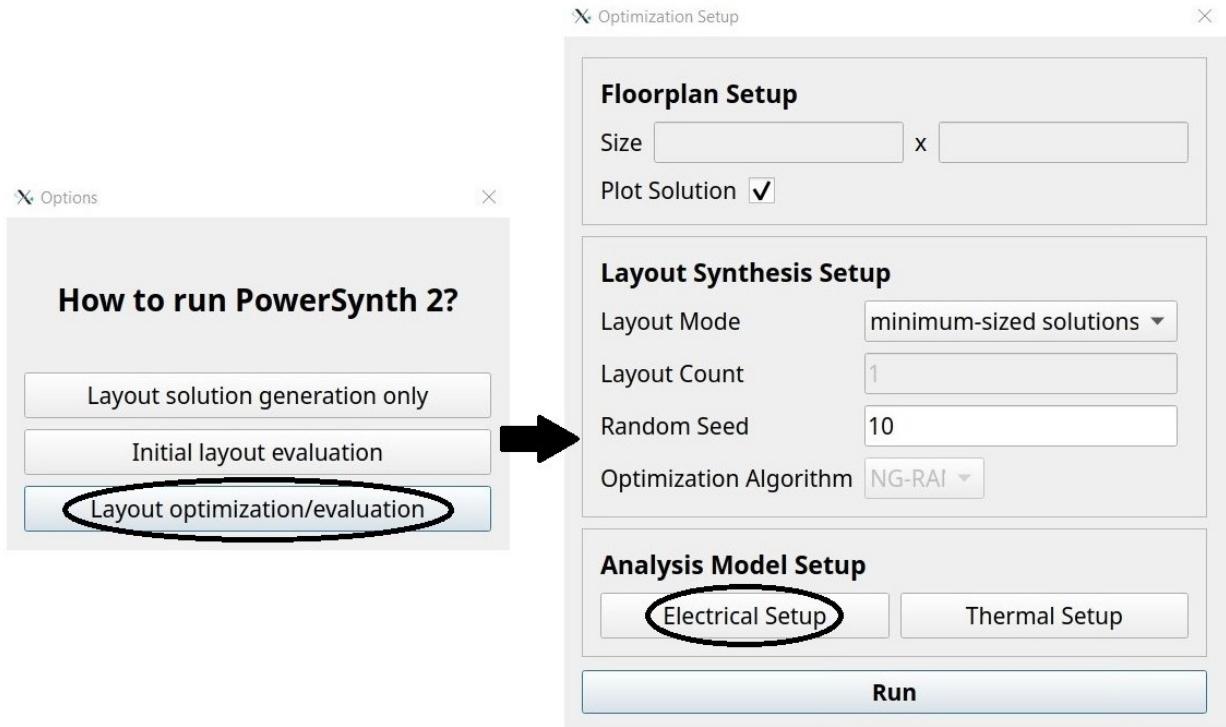


Figure 28: Start optimization

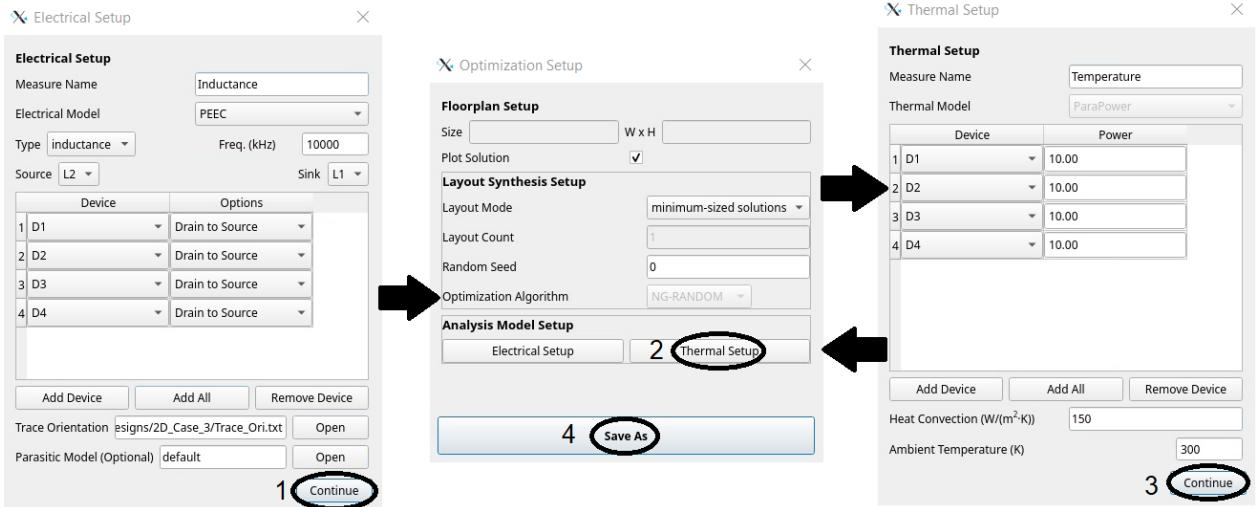


Figure 29: Optimization setup

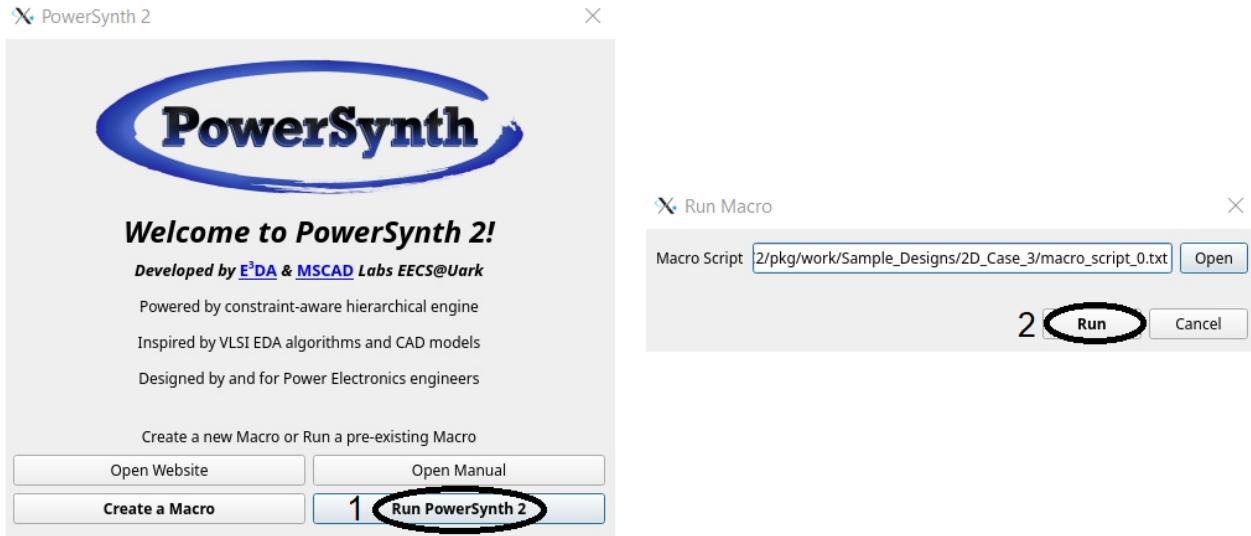


Figure 30: run PowerSynth optimization

15. On the thermal setup window, the thermal model is **ParaPower**. We need to provide a name of the measurement and then set up each device's heat dissipation value in Watts (W). Since we have four devices, we have added four devices and selected each device with corresponding heat dissipation at 10 W. Since it's a 2D module with single-sided cooling, we have provided a heat transfer coefficient of $150 \text{ W/m}^2\text{K}$. Finally, we have set up the ambient temperature at 300 K. Then, hitting the **Continue** button will take back to the **Optimization Setup** window. Since both electrical and thermal modeling set up are done, we can click on the **Save AS** to save the macro script. After that, by clicking **Run PowerSynth 2** button and **Run** button to evaluate the result. All of these steps are summarized in Figures 29, 29 .
16. After evaluation, the **Solution Browser** window will pop up (as shown in Figure 31). In this window, there will be a single in the solution space plot as we have generated a single solution. Upon clicking on the point, the corresponding layout will appear in the left window and performance values will be shown in the bottom of the window. From this window, the solution layout information can be exported in csv format by clicking either **Export Selected** or **Export All**. Also, to visualize the **initial layout**, the **Initial Layout** button can be pressed. This process has exported a **Macro script** in the project directory, which can be used for re-rerunning in different modes with **Run a Project** flow from the start GUI.

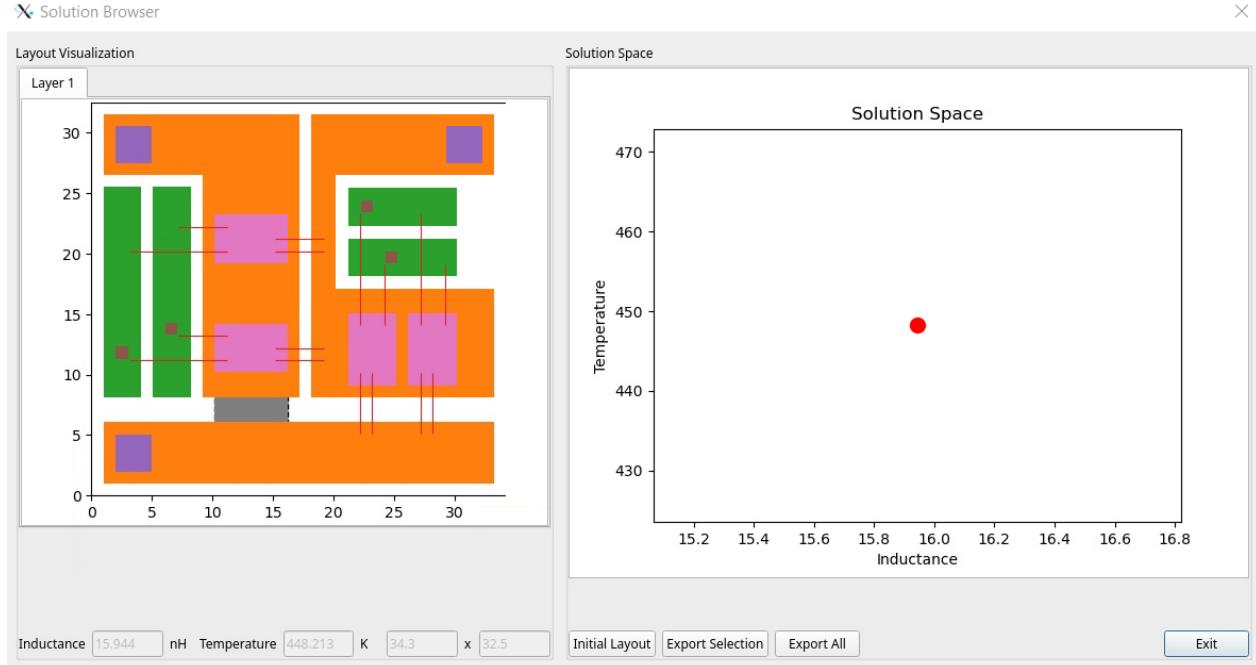


Figure 31: Solution browser

Since we have evaluated the minim-sized solution and got the performance values, and floorplan size, we can run PowerSynth to optimize the layout. To optimize, we can use the macro script that is saved in the project directory. We can modify the script and run PowerSynth in **Run PowerSynth 2** Mode. If the **macro_script** is opened, there will be no floorplan size in the macro script, as we did not enter any floorplan size in the previous run (**Create a Macro**). To optimize the layout for a given floorplan size, we need to add that information. We also need to enter the number of layouts we want to generate using **NG-RANDOM** optimization algorithm. If the user wants, the optimization algorithm can be changed to '**NSGAII**' or '**MOPSO**' which should add the number of generations as well. In this example, we want to optimize the layout for the floorplan size $40mm \times 50mm$ using **NG-RANDOM**, **NSGAII**, and **MOPSO**. the number of layouts is considered 400, and the number of generations (Iterations) is considered 9. So, the macro script for NSGAII should look like the following Listing 9. For NG-RANDOM and MOPSO we should change the NSGAII to them. The changes are: **Option**, **Layout_mode**, **Floor_plan**, **Num_of_layouts**, **Optimization_Algorithm**, **Num_generations**.

Listing 9: Macro script for optimization

```
# Input scripts:  
Layout_script: <path>/layout_geometry_script.txt  
Layer_stack: <path>/layer_stack.csv  
Parasitic_model: default  
Fig_dir: <path>/Figs  
Solution_dir: <path>/Solutions  
Constraint_file: <path>/constraint.csv  
Model_char: <path>/Characterization  
Trace_Ori: <path>/Trace_Ori.txt  
# Layout Generation Set up:  
Reliability-awareness: 0  
New: 0  
Plot_Solution: 1  
Option: 2  
Layout_Mode: 2  
Floor_plan: 40,50  
Num_of_layouts: 400  
Seed: 10  
Optimization_Algorithm: NSGAI  
Num_generations: 9  
  
Electrical_Setup:  
Model_Type: PEEC  
Measure_Name: Inductance  
Measure_Type: 0  
# Device Connection Table  
Device_Connection:  
D1 1,0,0  
D2 1,0,0  
D3 1,0,0  
D4 1,0,0  
End_Device_Connection.  
Source: L2  
Sink: L1  
Frequency: 10000  
End_Electrical_Setup.  
  
Thermal_Setup:  
Model_Select: 2  
Measure_Name: Temperature  
Selected_Devices: D1,D2,D3,D4  
Device_Power: 10,10,10,10  
Heat_Convection: 150  
Ambient_Temperature: 300  
End_Thermal_Setup.
```

To run the updated macro script, we need to run the executable and click on the **Run PowerSynth 2** button. Then the **macro_script** file location needs to be provided. The flow is shown in Figure 32. Then, once the **Run** button is clicked, the optimization will start and after waiting for a while, the solution browser will pop up and should look like the Figures 33, 34, and 35 for NG-RANDOM, NSGAII, and MOPSO respectively.

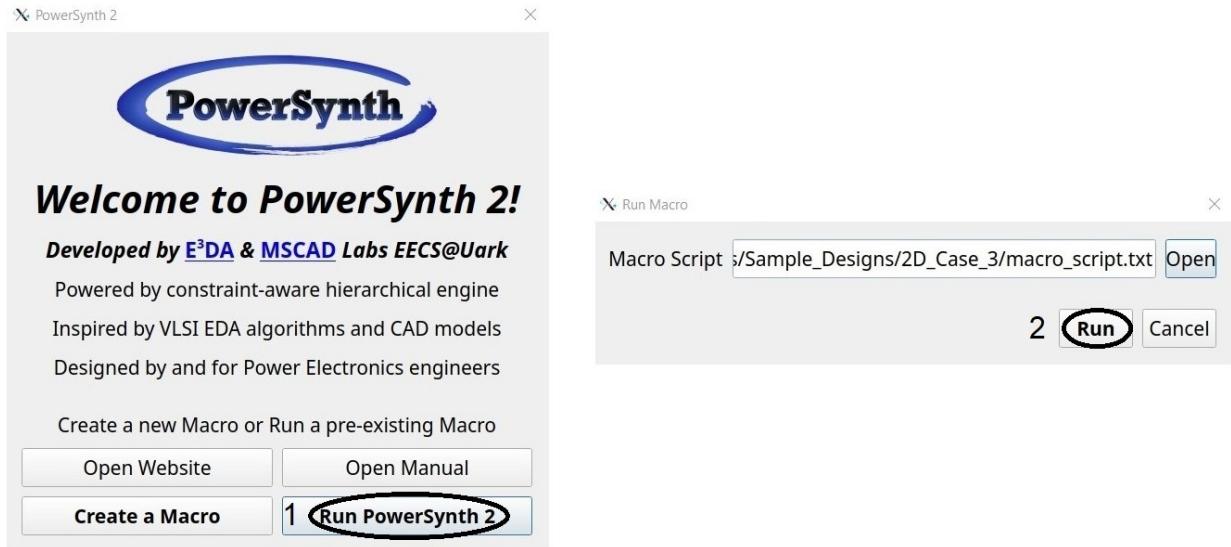


Figure 32: Run a Project Flow

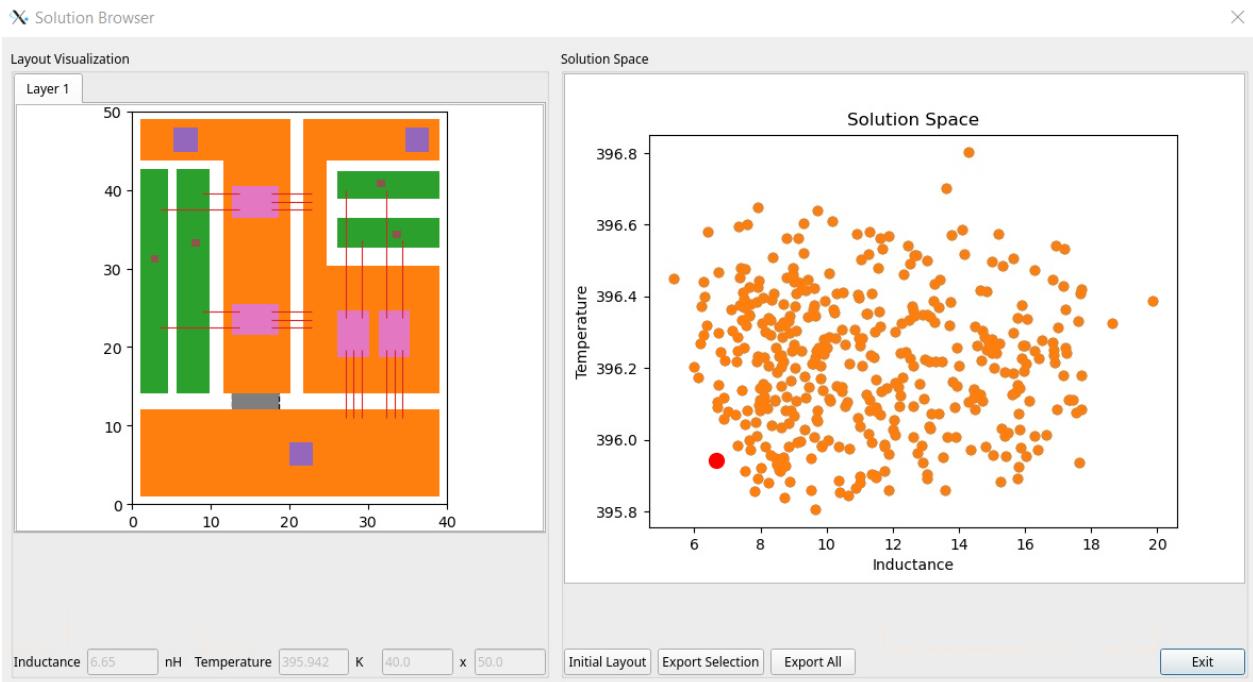


Figure 33: NG-RANDOM Solution space for 2D layout optimization



Figure 34: NSGAII Solution space for 2D layout optimization

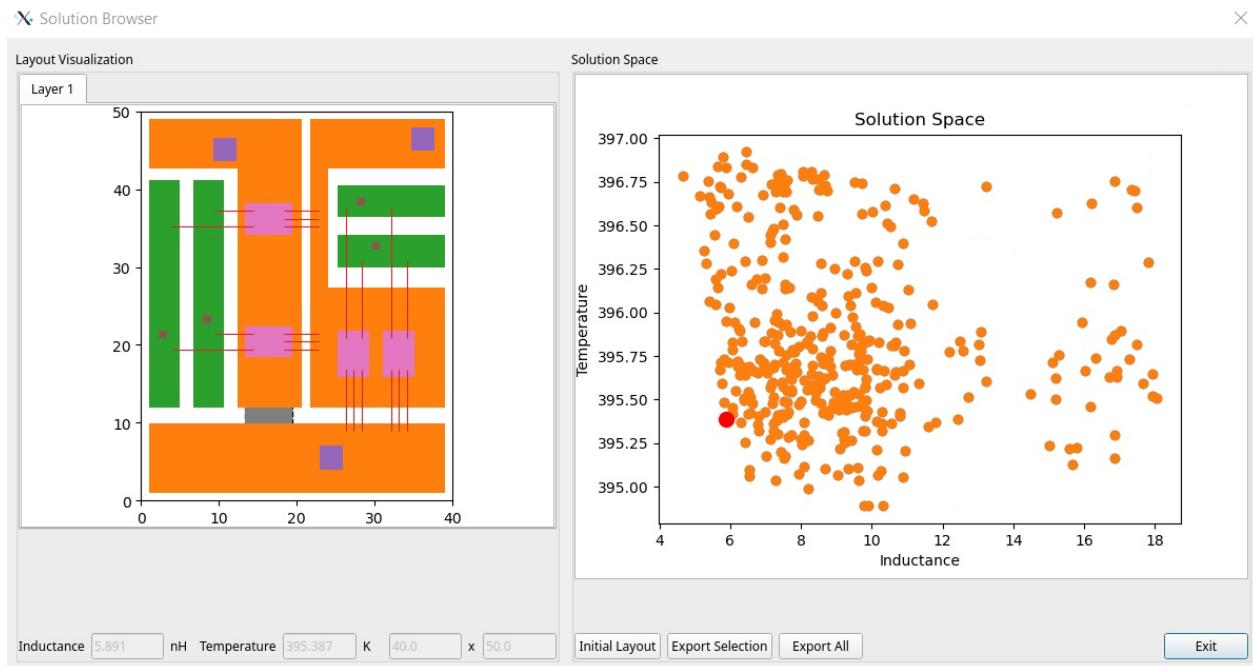


Figure 35: MOPSO Solution space for 2D layout optimization

2.4.2 3D MCPM Layout Optimization

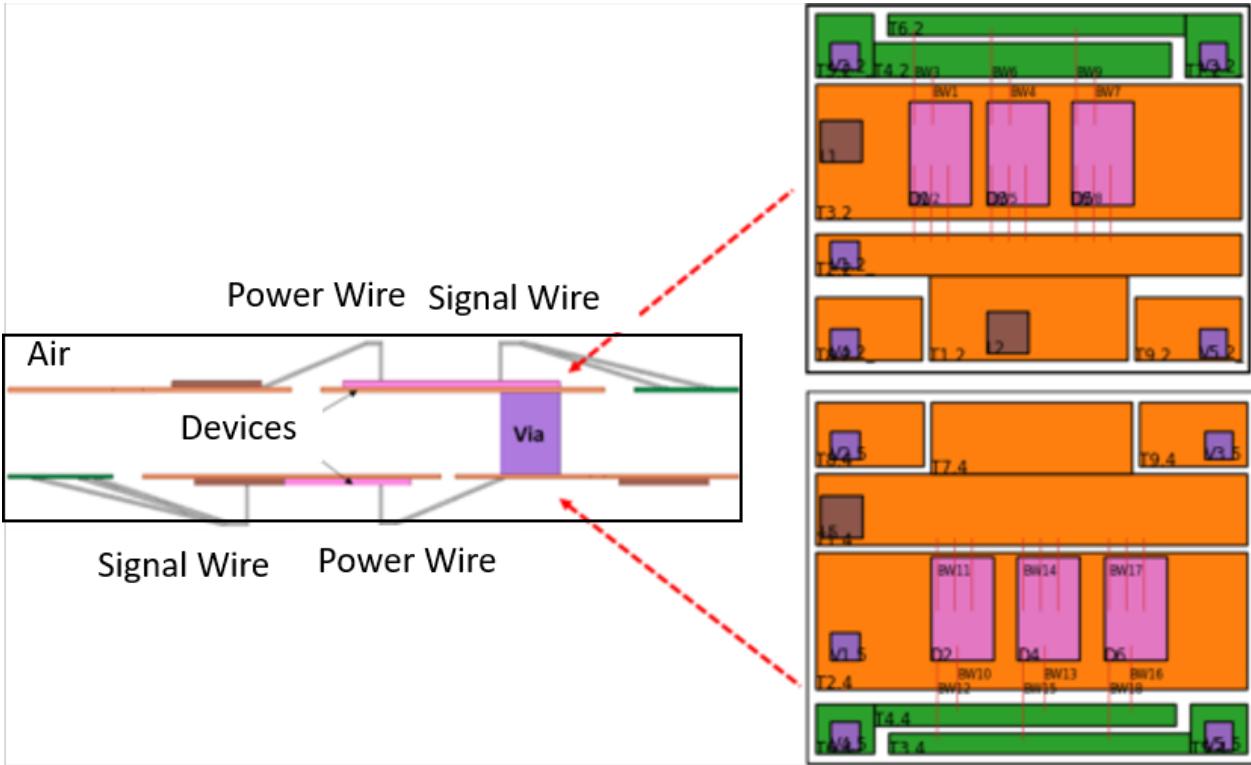


Figure 36: Initial 3D layout

To illustrate the 3D MCPM layout optimization capability, the initial layout shown in Figure 36. To optimize the power loop inductance from DC+ (L1)-to-DC- (L5) and maximum temperature of the layout, the layout script needs to be generated as shown in 11. Also, to represent the layout we need to have the following parts:

1. MOSFET, 2. Power Lead, 3. Signal Lead, 4. Capacitor. The steps are described as follows:

1. Prepare all part files and save in **Part_Lib** folder. So, this folder should have:
 1. MOSFET1.part (Content is shown in Listing 10)
 2. Power_Lead1.part (Content is shown in Listing 11)
 3. Vial1.part (Content is shown in Listing 12)

Listing 10: MOSFET1.part file content

```
Name CPM2-1200-0040B
Type component
Footprint 4.36 7.26
Thickness 0.18
Material SiC
Pins Drain Source Gate
  Drain 0 0 4.36 7.26 B
  Source 0.33 0.31 3.66 5.26 T
  Gate 1.68 5.67 0.84 0.6 T
Parasitics
  Drain Source R:1e-4
  Drain Gate R:1.1
  Gate Source R:1.1
```

Listing 11: power_lead1.part file content

```
Name PL1
Type connector
Footprint 3 3
Thickness 0.18
Material Copper
```

Listing 12: Via1.part file content

```
Name Via
Type connector
Footprint 2 2
Thickness 0.1
Material Cu
```

2. Prepare the wire definition file and save in **Wire_Lib** folder. So, this folder should contain the ‘bond_wire_info.wire’ file (content is shown in Listing 13).

Listing 13: bond_wire_info.wire file content

```
JEDEC-4 points
Resistivity 0.000001
Radius 0.127
```

3. Prepare layer stack file and save as .csv format. The file content for this design is shown in Table 5. This layer stack is different from the regular one to apply effective heat transfer coefficient on the top and bottom surface of the module. Since this is a wire bonded 3D module, to make an even surface on both sides, a layer of air has been considered surrounding the module to fill the gap and make an even surface on both sides.

Table 5: Layer_stack file content

ID	Name	Origin	Width	Length	Thickness	Material	Type	Electrical
1	C1	0,0	37.5	37.5	0.255	None	a	C
2	I1	0,0	37.5	37.5	0.1	copper	p	S
3	Ceramic1	0,0	37.5	37.5	0.5	Al_N	p	D
4	I2	0,0	37.5	37.5	0.1	copper	p	S
5	C2	0,0	37.5	37.5	0.255	None	a	C
6	Baseplate	0,0	37.5	37.5	1.21	Air	p	G

4. Create a blank csv file for constraints. For this case, the file name is "constraint.csv".
5. Prepare the trace orientation file. The content for this design is shown in Listing 14
6. Now, run PowerSynth executable and follow the steps as shown in 2D case to generate the macro script by running the minimum-sized solution generation flow. Once the macro script and constraint file are updated, the script will look like the listing 15. Here, we are generating a solution space with 200 solutions using **NG-RANDOM** optimization algorithm for floorplan size of $37.5mm \times 37.5mm$ (Layout mode 1). In addition, we run this example using **NSGAII** and **MOPSO**.

Listing 14: Trace_Ori file content

```
H : T2 . 2 , T3 . 2 , T4 . 2 , T1 . 4 , T2 . 4 , T3 . 4 , T4 . 4 , T6 . 2 , T9 . 4 , T8 . 4 , T8 . 2 , T9 . 2
V : T1 . 2 , T5 . 2 , T7 . 2 , T6 . 4 , T5 . 4 , T7 . 4
```

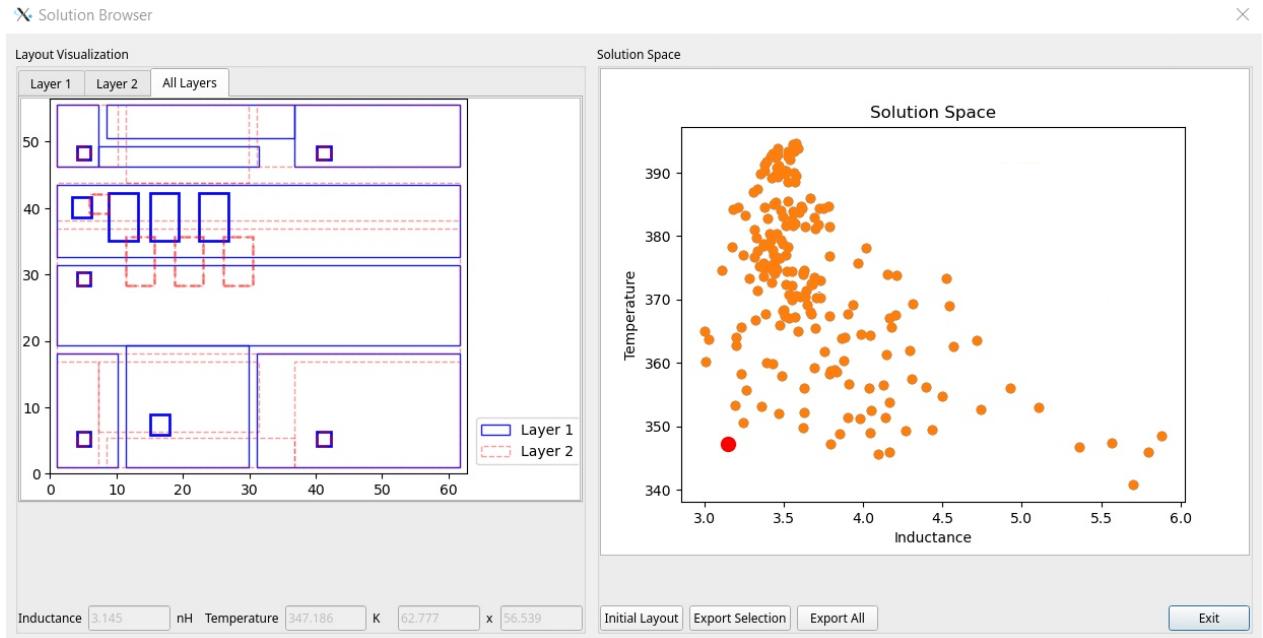


Figure 37: NG-RANDOM Solution space for 3D MCPM layout

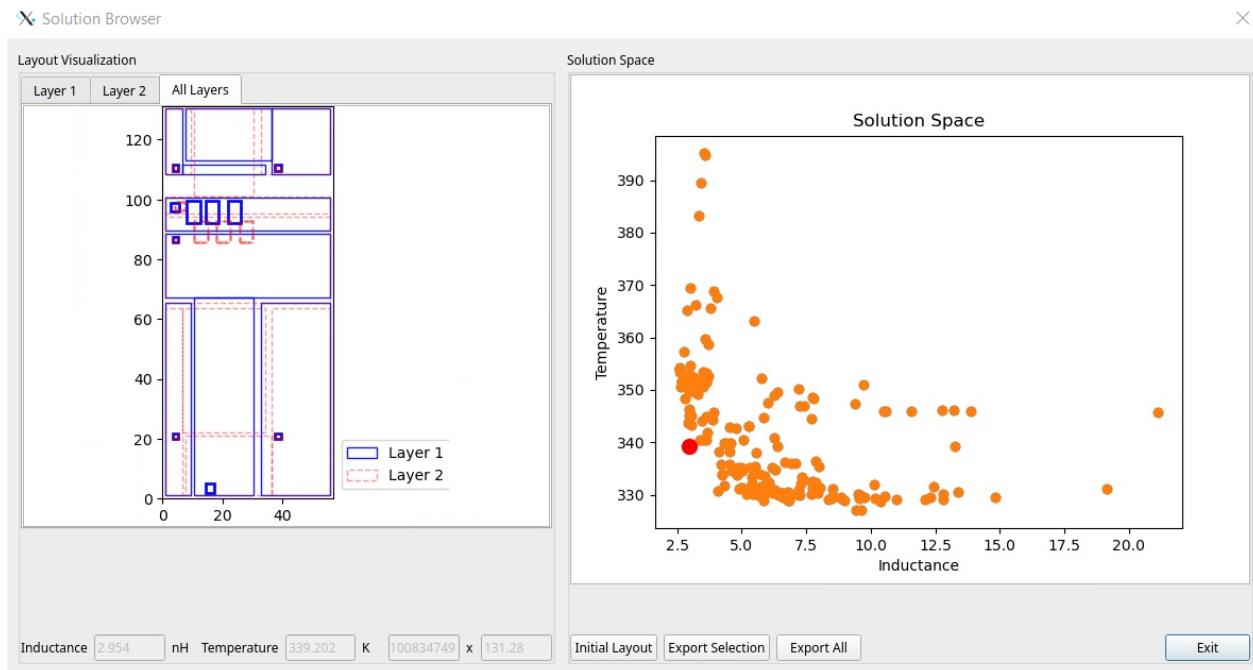


Figure 38: NSGAII Solution space for 3D MCPM layout

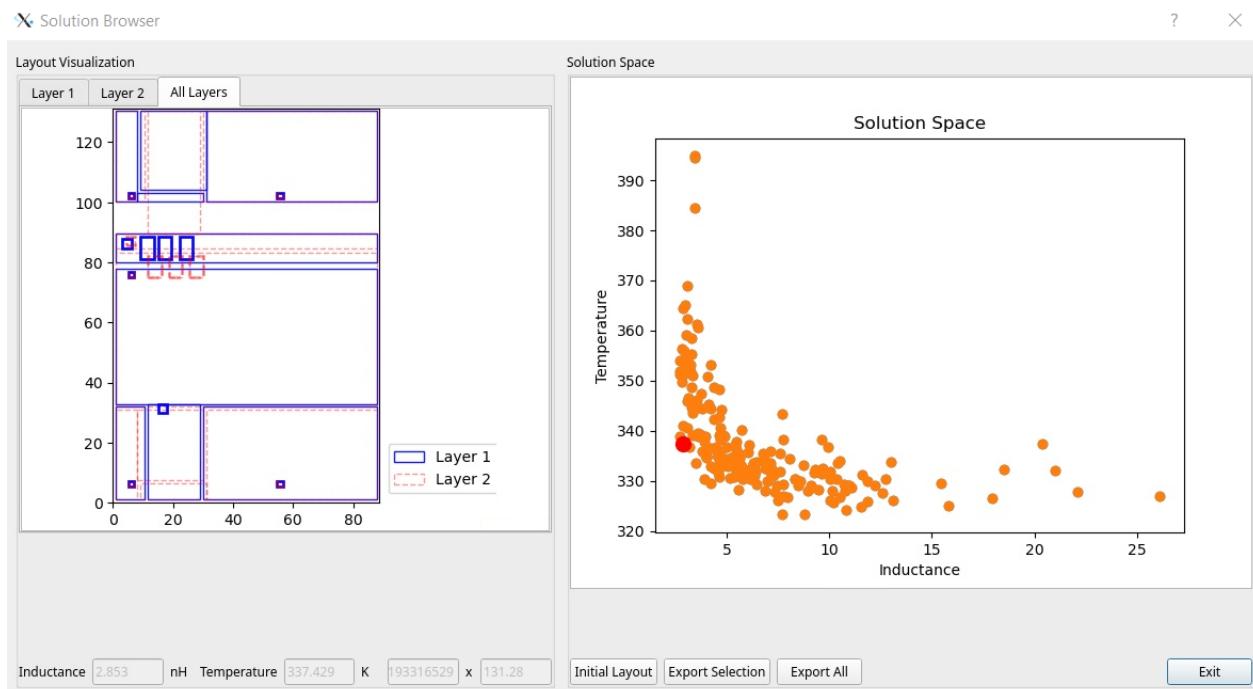


Figure 39: MOPSO Solution space for 3D MCPM layout

Listing 15: Macro script for optimization

```
# Input scripts:  
Layout_script: <path>/layout_geometry_script.txt  
Layer_stack: <path>/layer_stack.csv  
Parasitic_model: default  
Fig_dir: <path>/Figs  
Solution_dir: <path>/Solutions  
Constraint_file: <path>/constraint.csv  
Model_char: <path>/Characterization  
Trace_Ori: <path>/Trace_Ori.txt  
# Layout Generation Set up:  
Reliability-awareness: 0  
New: 0  
Plot_Solution: 1  
Option: 2  
Layout_Mode: 2  
Floor_plan: 37.5,37.5  
Num_of_layouts: 200  
Seed: 10  
Optimization_Algorithm: NG-RANDOM  
Num_generations: 4  
Electrical_Setup:  
Model_Type: FastHenry  
Measure_Name: Inductance  
Measure_Type: 1  
# Device Connection Table  
Device_Connection:  
D1 1,0,0  
D2 1,0,0  
D3 1,0,0  
D4 1,0,0  
D5 1,0,0  
D6 1,0,0  
End_Device_Connection.  
Source: L5  
Sink: L1  
Frequency: 1000 #kHz  
End_Electrical_Setup  
Thermal_Setup:  
Model_Select: 2  
Measure_Name: Temperature  
Selected_Devices: D1,D2,D3,D4,D5,D6  
#Device_Power: 10,10,10,10,10,10  
Device_Power: 2.5,2.5,2.5,2.5,2.5,2.5  
Heat_Convection: 350,350  
Ambient_Temperature: 300  
End_Thermal_Setup.
```

- Once the optimization is complete the solution space will look like the one shown in Figure 37, 38, and 39 for NG-RANDOM, NSGAII, and MOPSO respectively..

2.4.3 Converter Layout Optimization (ongoing work)

Designing and optimizing high-density converters is extremely challenging as it requires designers to have a broad knowledge of circuits, systems, packaging, manufacture, layout, testing, and cost control. Power converter designers are still relying on FEM tools to perform computational-intensive simulations one at a time and draw PCBs manually. The tedious and time-consuming approach is not well-documented and still relies on experienced engineers. The converter level requires new design features. Unlike power modules that rely on wire-bonding and thin devices, surface-mounted, and bulky passive components must be supported. In addition, new design optimization algorithms, the electrical model, and thermal metrics including static and transient junction temperature evaluation must be considered.

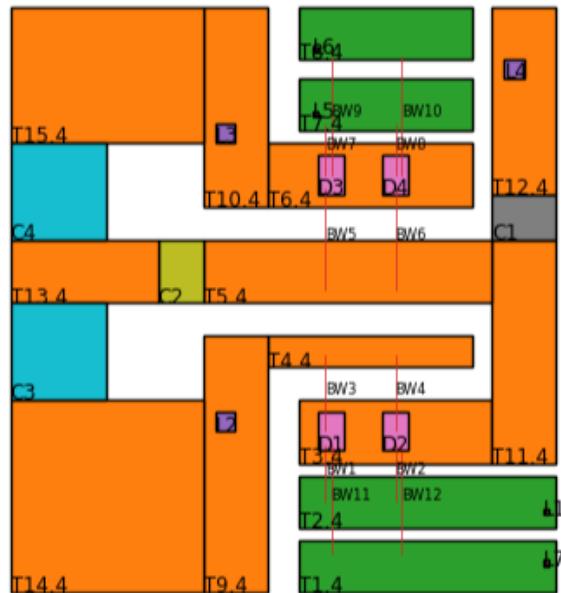


Figure 40: Initial Bidirectional dc-dc converter

To illustrate the converter layout optimization capability, the initial layout of a bidirectional dc-dc converter is shown in Figure 40. To optimize the power loop inductance from DC+ (L13)-to DC- (L2) and the maximum temperature of the layout, the layout script needs to be generated. Also, to represent the layout we need to have the following parts:

1. MOSFET, 2. Power Lead, 3. Signal Lead, 4. Capacitor, 5. Inductance. The steps are described as follows:

Prepare all part files and save them in **Part_Lib** folder. So, this folder should have:

1. MOSFET1.part (Content is shown in Listing 16)
2. Power_Lead1.part (Content is shown in Listing 2)
3. Signal_Lead.part (Content is shown in Listing 3)
4. capacitor.part (Content is shown in Listing 17)
5. inductance.part (Content is shown in Listing 18)

Listing 16: MOSFET.part file content

```
Name CPM2-1200-0025B
Type component
Link https://www.wolfspeed.com/downloads/dl/file/id/152/
product/2/cpm2\_1200\_0025b.pdf
Footprint 4 6
Thickness 0.18
Material SiC
Pins Drain Source Gate
  Drain 0 0 4 6 B
  Source 0.335 0.35 3.5 4.5 T
  Gate 1.03 5 0.5 0.5 T
Parasitics
  Drain Source R:25e-3
  Drain Gate R:1.1
  Gate Source R:1.1
```

Listing 17: capacitor.part file content

```
Name CKG57NX7T2J105M500JH
Type component
Footprint 15 15
Thickness 5
Material Al\_N
```

Listing 18: inductor.part file content

```
Name CKG57NX7T2J105M
Type component
Footprint 10 7
Thickness 5
Material Al\_N
```

After that, follow the steps as shown in the 2D case to generate the macro script by running the minimum-sized solution generation flow. Once the macro script and constraint file are updated, the script will look like the listing 19. Here, we are generating a solution space with 400 solutions using **MOPSO** optimization algorithm for Layout mode 1. Once the optimization is complete the solution space will look like the one shown in Figure 41.

Listing 19: Macro script for optimization

```
# Input scripts:  
Layout_script: <path>/layout_geometry_script.txt  
Layer_stack: <path>/layer_stack.csv  
Parasitic_model: default  
Fig_dir: <path>/Figs  
Solution_dir: <path>/Solutions  
Constraint_file: <path>/constraint.csv  
Model_char: <path>/Characterization  
Trace_Ori: <path>/Trace_Ori.txt  
# Layout Generation Set up:  
Reliability-awareness: 0  
New: 0  
Plot_Solution: 1  
Option: 2  
Layout_Mode: 1  
Floor_plan: 60,80  
Num_of_layouts: 400  
Seed: 10  
Optimization_Algorithm: MOPSO  
Num_generations: 9  
Electrical_Setup:  
Model_Type: FastHenry  
Measure_Name: Inductance  
Measure_Type: 1  
# Device Connection Table  
Device_Connection:  
D1 1,0,0  
D2 1,0,0  
D3 1,0,0  
D4 1,0,0  
End_Device_Connection.  
Source: L3  
Sink: L2  
Frequency: 1000 #kHz  
End_Electrical_Setup  
Thermal_Setup:  
Model_Select: 2  
Measure_Name: Temperature  
Selected_Devices: D1,D2,D3,D4  
Device_Power: 10,10,10,10  
Heat_Convection: 150  
Ambient_Temperature: 300  
End_Thermal_Setup.
```

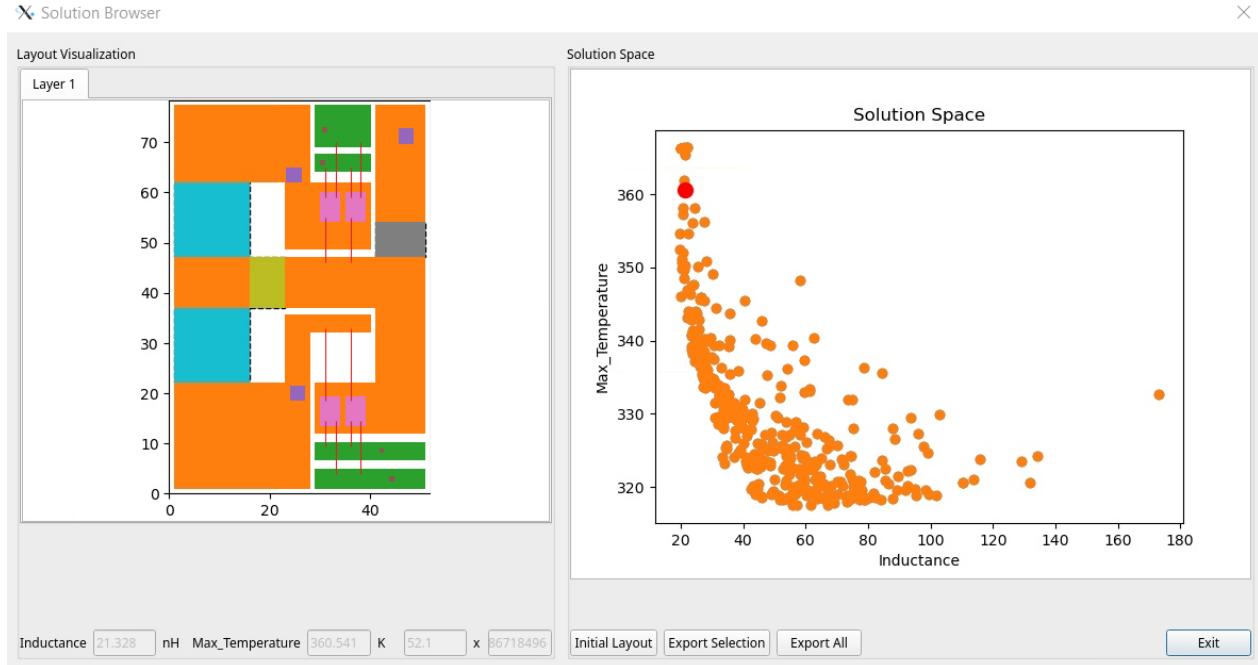


Figure 41: MOPSO Solution space for converter layout

More sample design cases are provided in the package (**Sample_Designs** folder). Those examples can help the user preparing own test cases.

3 PowerSynth-Related Publications

This section contains a list of all the current publications related to PowerSynth as of the date of this document.

1. P. Tucker, "SPICE netlist generation for electrical parasitic modeling of multi-chip power module designs," 2013.
2. B. W. Shook, A. Nizam, Z. Gong, A. M. Francis and H. A. Mantooth, "Multi-objective layout optimization for multi-chip power modules considering electrical parasitics and thermal performance," in *Control and Modeling for Power Electronics (COMPEL), 2013 IEEE 14th Workshop on*, 2013.
3. B. W. Shook, Z. Gong, Y. Feng, A. M. Francis and H. A. Mantooth, "Multi-chip power module fast thermal modeling for layout optimization," *Computer-Aided Design and Applications*, vol. 9, pp. 837-846, 2012.
4. B. W. Shook, "The Design and Implementation of a Multi-Chip Power Module Layout Synthesis Tool," 2014.
5. J. Main, "A Manufacturer Design Kit for Multi-Chip Power Module Layout Synthesis," 2017.
6. Q. Le, T. Evans, S. Mukherjee, Y. Peng, T. Vrotsos and H. A. Mantooth, "Response surface modeling for parasitic extraction for multi-objective optimization of multi-chip power modules (MCPMs)," in *Wide Bandgap Power Devices and Applications (WiPDA), 2017 IEEE 5th Workshop on*, 2017.

7. Q. Le, S. Mukherjee, T. Vrotsos and H. A. Mantooth, "Fast transient thermal and power dissipation modeling for multi-chip power modules: A preliminary assessment of different electro-thermal evaluation methods," in *Control and Modeling for Power Electronics (COMPEL), 2016 IEEE 17th Workshop on*, 2016.
8. Z. Gong, "Thermal and electrical parasitic modeling for multi-chip power module layout synthesis," 2012.
9. S. Mukherjee et al, "Toward Partial Discharge Reduction by Corner Correction in Power Module Layouts," in *Control and Modeling for Power Electronics (COMPEL)*, pp. 1–8, Jun 2018.
10. I. Al Razi et al, "Constraint-Aware Algorithms for Heterogeneous Power Module Layout Synthesis and Reliability Optimization," in *Wide Bandgap Power Devices and Applications (WiPDA)*, pp. 323–330, Oct 2018.
11. T. Evans, Q. Le, S. Mukherjee, I. Al Razi, T. Vrotsos, Y. Peng and H. A. Mantooth, "PowerSynth: A Module Layout Generation Tool," in *IEEE Transactions on Power Electronics*, vol. 34, no. 6, pp. 5063–5078, Jun 2019, Highlighted Paper.
12. Q. Le et al, "PEEC Method and Hierarchical Approach Towards 3D Multichip Power Module (MCPM) Layout Optimization", in Proc. IEEE International Workshop on Integrated Power Packaging, pp. 131–136, Apr 2019.
13. I. Al Razi et al, "Hierarchical Layout Synthesis and Design Automation for 2.5D Heterogeneous Multi-Chip Power Modules", in Proc. IEEE Energy Conversion Congress and Exposition, pp. 2257-2263, Sep 2019.
14. T. Evans et al, "Development of EDA Techniques for Power Module EMI Modeling and Layout Optimization", in Proc. IMAPS International Symposium on Microelectronics, pp. 193-198, Oct 2019.
15. Y. Peng et al, "PowerSynth Progression on Layout Optimization for Reliability and Signal Integrity", IEICE Nonlinear Theory and Its Applications, vol. 11, no. 2, pp. 124-144, Apr 2020, Invited Paper.
16. I. Al Razi et al, "Physical Design Automation for High-Density 3D Power Module Layout Synthesis and Optimization", in Proc. IEEE Energy Conversion Congress and Exposition, pp. 1984–1991, Oct 2020..
17. T. Evans et al, "Electronic Design Automation Tools and Considerations for Electro-Thermo-Mechanical Co-Design of High Voltage Power Modules", in Proc. IEEE Energy Conversion Congress and Exposition, pp. 5046–5052, Oct 2020.
18. S. Mukherjee et al, "General Equation to Determine Design Rules for Mitigating Partial Discharge and Electrical Breakdown in Power Module Layouts", in Proc. IEEE Workshop on Wide Bandgap Power Devices and Applications in Asia, pp. 1–6, Sep 2020.
19. I. Al Razi et al, "PowerSynth-Guided Reliability Optimization of Multi-Chip Power Module", in Proc. IEEE Applied Power Electronics Conference, pp. 1516-1523, Jun 2021.
20. Q. Le et al, "PowerSynth Integrated CAD Flow for High Density Power Modules", in Proc. IEEE Design Methodologies Conference, pp. 1-6, Jul 2021.

21. T. Evans et al, "Placement and Routing for Power Module Layout", in Proc. IEEE Design Methodologies Conference, pp. 1-6, Jul 2021.
22. J. Mitchener et al, "Designing a Graphical User Interface for the Power Module Optimization Tool PowerSynth", in Proc. ASEE Midwest Section Conference, pp. 1-12, Sep 2021.
23. Q. Le et al, "Fast and Accurate Inductance Extraction For Power Module Layout Optimization Using Loop-Based Method", in Proc. IEEE Energy Conversion Congress and Exposition, pp. 1358-1365, Oct 2021.
24. I. Al Razi et al, "Hierarchical Layout Synthesis and Optimization Framework for High-Density Power Module Design Automation", in Proc. International Conference on Computer-Aided Design, pp. 1-8, Nov 2021.
25. I. Al Razi et al, "PowerSynth Design Automation Flow for Hierarchical and Heterogeneous 2.5D Multi-Chip Power Modules", IEEE Transactions on Power Electronics, vol. 36, no. 8, pp. 8919–8933, 2021.
26. Q. Le et al, "Thermal Runaway Mitigation through Electrothermal Constraints Mapping for MCPM Layout Optimization", in Proc. IEEE Design Methodologies Conference, pp. 1-6, Sept 2022.
27. I. Al Razi et al, "Electromigration-Aware Reliability Optimization of MCPM Layouts Using PowerSynth", in Proc. IEEE Energy Conversion Congress and Exposition, pp. 1-8, Oct 2022.
28. Q. Le et al, "Fast and Accurate Parasitic Extraction in Multichip Power Module Design Automation Considering Eddy-Current Losses", IEEE Journal of Emerging and Selected Topics in Power Electronics, 2022.
29. I. Al Razi et al, "PowerSynth 2: Physical Design Automation for High-Density 3D Multi-Chip Power Modules", IEEE Transactions on Power Electronics, vol. 38, no. 4, pp. 4698-4713, 2023.
30. M. Sanjabiasasi et al, "A Comparative Study on Optimization Algorithms in PowerSynth 2", Proc. IEEE Design Methodologies Conference, Sep 2023.
31. Z. Saadatizadeh et al, "Automated Layout Optimization Methods of a Bidirectional DC-DC ZVS Converter Using PowerSynth", Proc. IEEE Design Methodologies Conference, Sep 2023.

3.1 Useful Links

- **Release Website:** <https://e3da.csce.uark.edu/release/PowerSynth/>
- **Publication Website:** <https://e3da.csce.uark.edu/pub/>

4 Authors

The PowerSynth tool development has been ongoing for more than a decade now and the current PowerSynth team appreciates efforts from quite a few number of graduate students and numerous undergraduate students over the years. We are grateful to Yugo Isogai, an undergrad from Electrical Engineering Department for his effort on the MDK editor GUI development and REU student Joshua Mitchener for his effort on the PowerSynth 2.0 GUI development. In addition, we would like to thank Dr. Zahra Saadatizadeh, and David Agogo-Mawuli, for their contributions on PowerSynth 2.1.

4.1 Graduate Research Assistants/ Graduates

The PowerSynth research and development team worked on this version (v2.1) consisting of two recent graduates and a Ph.D. student. A brief introduction to the authors are as follows:

- Mehran Sanjabiasasi**

Ph.D. Student in Computer Engineering, 2023

Electrical Engineering and Computer Science Department
University of Arkansas, Fayetteville, AR, USA.

- Imam Al Razi**

Ph.D. in Computer Engineering, 2022

Computer Science and Computer Engineering Department
University of Arkansas, Fayetteville, AR, USA.

- Quang Le**

Ph.D. in Electrical Engineering, 2022

Electrical Engineering Department
University of Arkansas, Fayetteville, AR, USA.

- Tristan M. Evans**

Ph.D. in Electrical Engineering, 2023

Electrical Engineering Department
University of Arkansas, Fayetteville, AR, USA.

- Shilpi Mukherjee**

Ph.D. in Microelectronics and Photonics, 2023

Microelectronics and Photonics Department
University of Arkansas, Fayetteville, AR, USA.

4.2 Supervisors

- Dr. Homer Alan Mantooth**

Distinguished Professor, The Twenty-First Century Research Leadership Chair in Engineering
Department of Electrical Engineering, University of Arkansas, Fayetteville, AR, USA.

Phone: 479-575-4838

Email: mantooth@uark.edu

- Dr. Yarui Peng**

Assistant Professor

Computer Science and Computer Engineering Department
University of Arkansas, Fayetteville, AR, USA.

Phone: 479-575-6043

Email: yrpeng@uark.edu