The Design and Implementation of a Multi-Chip Power Module Layout Synthesis Tool

The Design and Implementation of a Multi-Chip Power Module Layout Synthesis Tool

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering

by

Brett W. Shook
University of Arkansas
Bachelor of Science in Electrical Engineering

May 2014
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____
Dr. H. Alan Mantooth
Thesis Director

_____
Dr. Simon Ang
Committee Member

_____
Dr. Randy Brown
Committee Member

ABSTRACT

This thesis presents the design and implementation of a software tool, referred to as PowerSynth, which is used to synthesize multi-chip power module (MCPM) layouts and also aids in the overall design process. MCPMs allow power electronics systems to achieve higher efficiencies and reduced size, weight, and cost through the compact integration of devices into a single package. MCPM design is highly multidisciplinary which involves electrical, thermal, and mechanical performance considerations. Conventionally, the design process may require several iterations and multiple software modeling tools to create high performance MCPMs. The layout of an MCPM consists of a pattern of metal traces and the placement of components on these traces. The design of an MCPM layout and the selection of materials, dimensions, and components used in an MCPM affect its performance. PowerSynth incorporates thermal and electrical models into a single tool and provides a unified system for designing MCPMs and synthesizing layouts for them. A user is guided through the design process with graphical user interfaces and is able to select from a set of solutions which trade-off thermal and electrical performance. Final MCPM designs can be exported to two different commercial modeling tools for further performance verification or for manufacturing artwork generation.

ACKNOWLEDGEMENTS

The process of writing this piece of software has taught me an immense amount about a diverse set of topics such as finite element analysis, thermal and electrical modeling, optimization, software development, etc. I would like to thank Dr. Mantooth for setting me on this journey through such a wide range of material and his leadership throughout the project. Dr. Matt Francis has mentored me meeting by meeting every week for several years helping to dispel uncertainty and promote understanding. Dr. Yongfeng Feng also mentored me throughout the beginning of this project and spent many late nights working out problems with me. This work has been made possible by the GRid-connected Advanced Power Electronic Systems (GRAPES) NSF research center. The industry advisory board of this organization has provided me many insights into what is truly needed to improve the engineering of power electronics. Brice McPherson and Jack Bourne at Arkansas Power Electronics International (APEI) provided key information on the design process of MCPMs and many interesting discussions.

Zihao Gong, my initial partner on this project, always kept a positive attitude and spent countless hours working with me to develop the tool. Undergraduates Peter Tucker and Seth Mayfield did a prodigious job helping put together the user interfaces and several extremely useful export functions. Matt Barlow provided great insight into his technical challenges regarding power module design. Dr. Simon Ang, Dr. Shirley Zhang, and Atanu Dutta provided the project with real world module layouts and data which was immensely helpful. Dr. Randy Brown taught me much throughout my degree and guided me through the process. Kathy Kirk has worked tirelessly to help facilitate communication between all parties. I would like to thank everyone mentioned above and anyone else who has helped me through this process.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDIX FIGURES

LIST OF TABLES

Chapter 1 Introduction

1.1 Background and Motivation

Power electronics modules are used to control the flow of electrical energy to or from electrical systems such as batteries, electric motors, the grid, renewable energy sources, etc. They are responsible for large increases of efficiency in manufacturing and many other areas involving energy transfer. An ideal power module is low cost, small in size, highly efficient, and highly reliable.

These goals are primarily achieved by increasing the switching frequency at which a module operates. The switching frequency determines how fast the semiconductor devices inside the power module are modulated to control the flow of power. At higher switching frequencies the passives such as capacitors, inductors, and high frequency transformers connected to the module can generally be reduced in size. This reduces the cost of the overall system and increases efficiency [1]. However, as the switching frequency is increased past some threshold, depending on the module parasitics and semiconductor devices, the switching losses eventually overcome the efficiency gains.

In order to increase the switching frequency of power modules, layout parasitics need to be reduced. A power module layout is the pattern of metal traces and placement of components on the traces. Parasitic inductances in the layout of a module can cause over-voltages across semiconductor devices and EMI issues which can cause reduced operating lifetime or even catastrophic failure of devices, see Fig. 1.1.

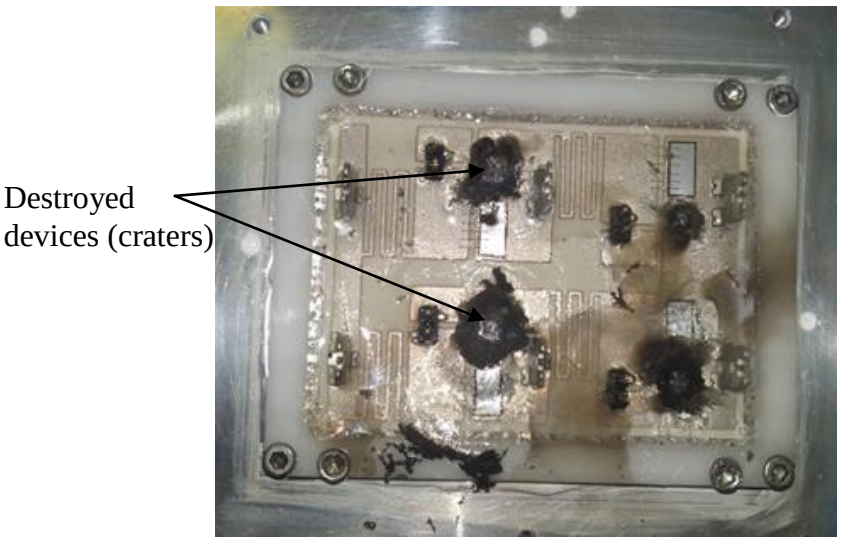


Fig. 1.1: Catastrophic failure of an MCPM.

Multi-chip power modules (MCPMs), which are the focus of this thesis, help reduce layout parasitics by integrating multiple bare die power devices and control circuitry into a single package [2]. Integration helps reduce the conductor path lengths between devices, thus reducing parasitic inductance. As devices are integrated into smaller packages, thermal design considerations become important.

Semiconductor devices operating at high frequency may generate up to several watts of waste heat which flows into the surrounding environment. Reducing the size of an MCPM increases heat flux density and device temperature which reduces device operating lifetime and reliability [3]. The heat transfer capability of a module is increased when its volume is increased, because of increased surface area. Spacing devices further from one another also decreases device temperature. The objective of reducing device temperatures is in conflict with the objective of reducing layout parasitics. As a module is decreased in size to decrease parasitics,

the device temperatures increase. Finding the correct trade-off between these two characteristics depends on the application of an MCPM, its intended environment, materials, and components.

The current design process of MCPMs is primarily done manually, although some experimental software tools have been developed to help automate the design problem. An automatic layout design tool for power modules which focuses on minimizing loop inductance and footprint area is presented in [4]. This tool does not consider the temperature of the devices in the module and only considers a single layout parasitic measure (loop inductance). The tool is only able to produce half-bridge topologies. It would be useful for an MCPM layout synthesis and design tool to consider both electrical and thermal performance criteria and other circuit topologies.

A second tool, a predecessor to the tool described in this thesis, models both layout inductance, resistance, and device temperatures [5]. This tool, referred to as PowerCAD, does not consider the trade-off between electrical parasitics and temperature, but treats the maximum temperature in the module as a constraint. A user may need to manually change the thermal constraint multiple times in order to find the correct trade-off between the desired thermal and electrical performance. PowerCAD also only provides support for half-bridge power modules which are configured in a specific ring shaped layout. The conclusions gained from building PowerCAD provided some key insights for building the current iteration of the tool described in this thesis.

The tool presented in thesis, PowerSynth, allows a user to consider the trade-off between thermal and electrical performance and also provides a flexible framework for synthesizing not only half-bridge MCPMs but any circuit topology. The tool is able to generate a set of optimal

trade-off MCPM designs considering any number of thermal or electrical performance criteria. PowerSynth includes graphical user interfaces which aid in the problem definition process and browsing of a solution set. A design is selected from this set and is easily exported to two different commercial modeling tools that subsequently allow generation of manufacturing artwork for fabrication.

1.2 MCPM Nomenclature and Spatial Orientation

An MCPM is composed of several different layers and components as shown in Fig. 1.2. The bottommost layer is a baseplate or heat spreader which helps dissipate waste heat from the semiconductor devices at the top of the module. A substrate layer resides on top of the baseplate and is formed out of a dielectric isolation layer which is surrounded by two metal layers. It is bonded to the baseplate by the substrate attach layer which is commonly composed of a solder type material. The dielectric layer of the substrate is usually composed of a ceramic material with high thermal conductivity. The two metal layers are usually composed of either copper or aluminum. The substrate may be referred to as direct bond aluminum (DBA) or direct bond copper (DBC), because the metal layers are directly bonded to the ceramic layer without a separate attachment compound. The metal layers are relatively thick compared to lower power circuit boards which allow them to tolerate high voltages and high currents. The top metal layer is etched to form a pattern of interconnection for the desired layout topology [2].

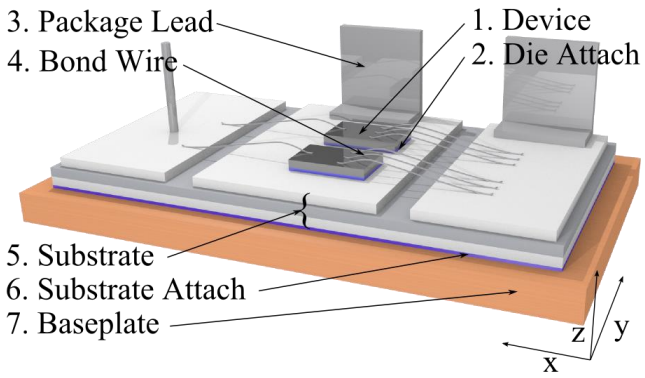


Fig. 1.2: Parts of an MCPM.

The power semiconductor devices used in MCPMs are usually un-packaged or bare die. The devices are connected to the metal traces by bondwires and by bonding the bottom surface of the die to a trace. The die attach or material that bonds the die to the metal surface is commonly a solder material. Package leads are bonded to the metal trace layer and provide either high power connections to the module or low power signal connections to the control electronics which are commonly integrated at the top. The control electronics may also be integrated onto a separate control substrate which is bonded to the power substrate, but this type of configuration is not targeted in this thesis [2].

The z direction is referred to as the vertical direction throughout the thesis. The thickness dimension runs in the vertical direction while width and length dimensions run in either the x or y directions depending on context.

A close-up image of a bare die metal oxide semiconductor field effect transistor (MOSFET) device commonly used in MCPMs is shown in Fig. 1.3. The device is a vertical MOSFET rather than a horizontal or planar MOSFET used in lower power circuitry. Current

flows through a vertical MOSFET from the bottom (drain) to the top (source). The drain is connected to the metal traces with a conductive die attach. Bondwires connect the source and gate pads at the top of the device. Insulated gate bipolar transistors (IGBTs) follow a similar connection pattern, but the terms base, emitter, and collector are used instead of gate, source, and drain for the input ports. IGBT type devices are not included in the tool described in this thesis, but integration for them would be relatively straight forward.

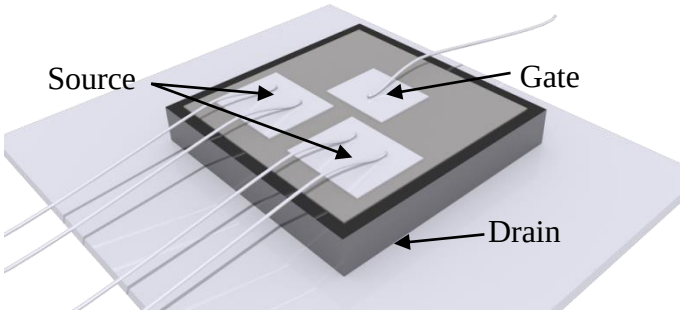


Fig. 1.3: Bare die vertical MOSFET.

Fast switching Schottky diodes are commonly found in MCPM layouts as bare die. The diode die are bonded to the substrate in the same way as MOSFET die. The bottom of a diode die is the cathode and the top pad is the anode. The anode pad on the top of a diode die is wirebonded in the same way as the source or gate connections for a MOSFET die.

The MOSFETs used in MCPMs are usually composed of either silicon (Si) or silicon-carbide (SiC). SiC MOSFETs are becoming more commonly used in MCPMs and other power electronics applications due to their relatively higher thermal conductivity, higher temperature operation, wider band gap, and fast switching rate as compared to Si based MOSFETs. SiC Schottky diodes have already become commonplace in power electronics designs.

Another term used frequently in reference to MCPMs and power electronics in general is loop inductance. The loop inductance is the amount of parasitic stray inductance in a path or loop in which the current is forced to stop flowing periodically. For example, two half-bridge MCPMs may be connected on either side of a load which is being supplied with an alternating current, see Fig. 1.4(a). The transistors M1 and M4 are turned on in unison to produce a current flowing from left to right through the load. Eventually, this current will be stopped and reversed or commutated by turning off M1 and M4 and turning on M2 and M3. While M1 and M4 are turning off, the current will begin to decrease and voltage will begin rise on the parasitic inductances represented by L_1 and L_4. If these parasitic inductances are large and the turn-off time is short, a large voltage will form across each inductance which can stress devices and/or decrease switching efficiency [6].

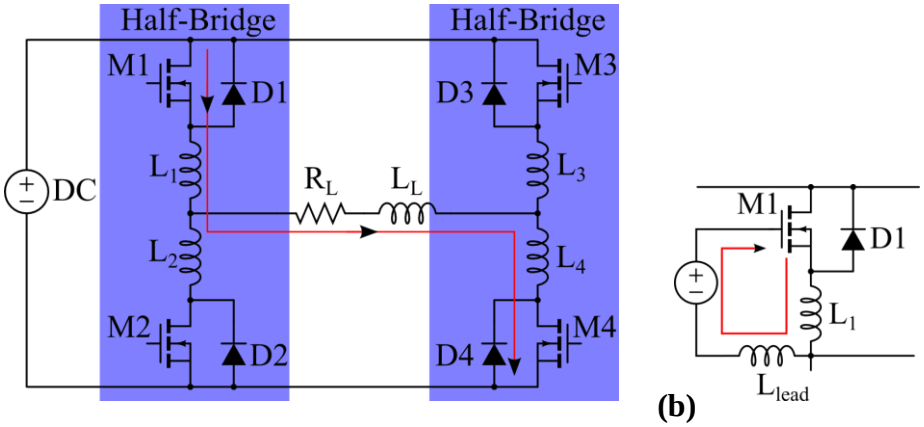


Fig. 1.4: Loop Inductance. (a): Main switching loop inductance with current path marked. (b): Gate loop inductance with current path marked.

The main switching loop inductance for a half-bridge module may be defined as the sum of all of the parasitic stray inductances from the positive to negative terminals such as $L_1 + L_2$ in the half-bridge on the right in Fig. 1.4(a). Gate loop inductance is also a frequently used term. As transistors are turned on and off during switching, a current appears in the path formed between

the gate and the source of the transistor as shown in Fig. 1.4(b). Minimizing this parasitic inductance may also help reduce ringing effects during switching states of the power transistor [6].

1.3 Thermal and Electrical Modeling

Thermal and electrical models are needed in order to evaluate the performance of MCPM designs. Finite element method (FEM) based thermal and electrical models are accurate but computationally expensive and time consuming to evaluate. FEM models are useful for the verification of a final design. PowerSynth uses an optimization algorithm to find many optimal MCPM designs and evaluates thousands of candidate designs in the process. FEM models are generally too slow for this optimization process. Fast thermal and electrical models are needed to perform the optimization process within a reasonable time frame e.g. 5 to 10 minutes.

The fast thermal modeling technique developed specifically for this tool is presented in detail in [7], [8]. The thermal model requires a characterization step to be carried out on each type of semiconductor device in the system. This characterization step provides steady state temperature and heat flux distributions for a device dissipating a constant heat flow. If multiple instances of a device exist in an MCPM, these distributions are placed in superposition. As the device instances are moved around during the optimization process, the device temperatures may be quickly recalculated through superposition. This characterization step has been automated and is presented in this thesis, thus removing any manual FEM modeling by a user.

The electrical parasitic modeling and extraction system is presented in [8], [9]. These works provide information about the physical closed-form equations used to model the parasitic

inductance, resistance, and capacitance of traces in an MCPM layout. These works only describe a parasitic extraction system for a fixed layout configuration and not a full parasitic extraction implementation. A more general parasitic extraction method is presented and tested in this thesis.

1.4 Optimization

An optimization process seeks to minimize or maximize a characteristic of a system by varying a set of parameters or variables which define the design of a system. In the case of MCPMs, there are multiple characteristics which are desired to be minimized e.g. maximum module temperature and loop inductance. The objective is to minimize both of these characteristics, but both cannot be minimized simultaneously. As the MCPM design parameters are changed to decrease the maximum module temperature, the loop inductance will increase. A trade-off exists between these two objectives. A multi-objective optimization process seeks to find a set of solutions which provide the best trade-off between all of the objectives. A designer is able to select a solution which best suits the needs of a particular engineering problem. For example, a designer may desire lower temperature operation for improved reliability, but a higher loop inductance which reduces electrical performance.

PowerSynth allows for definition of any number of objectives and is not limited to two. A designer may be interested in minimizing gate loop inductance or minimizing the temperature differences between devices in a layout in addition to reducing maximum temperature and main switching loop inductance.

1.5 Symbolic Layout System

A symbolic layout system is developed to provide a framework for defining MCPM layouts without specifying the scale and dimensions of trace or the positions of components. This abstraction allows a layout configuration to be used for multiple projects which have different devices, leads, bondwires, and geometric scales. A vector or list of design variables are extracted from a symbolic layout and passed to the optimizer. The optimizer varies these design variables in order to find optimal layout solutions. The symbolic layout system also forms the groundwork for a fully automated layout synthesis system which would generate layouts from circuit netlists. This goal has not yet been achieved in PowerSynth, but some steps have been taken towards it. An algorithm has been developed to produce symbolic layouts for half-bridge topologies, but this work is still in an experimental phase.

1.6 Software Design and Integration

The thermal and electrical models, multi-objective optimizer, and symbolic layout system are integrated together into a single tool using a set of data structures and graphic user interfaces (GUIs). The overall goal of the software is to reduce the amount of time and effort required to design an MCPM and layout. A non-expert in MCPM design should be able to use PowerSynth to design and export an MCPM. The software should also allow for faster design iteration time if multiple design iterations are needed.

A simplified software flowchart for PowerSynth is shown in Fig. 1.5. A user selects the materials, dimensions, and properties of the baseplate and substrate for an MCPM design. The baseplate and substrate materials are chosen from a technology library which holds material

property information and other data for use on an inter-project basis. A symbolic layout is loaded into the tool and specific device, lead, and bondwire types are chosen from the technology library to represent certain symbolic elements in the layout. Next, a user chooses a set of performance criteria which are critical to the design of the module such as maximum device temperature, loop inductance, etc. Pre-loaded projects may be saved for users with less experience in the design process.

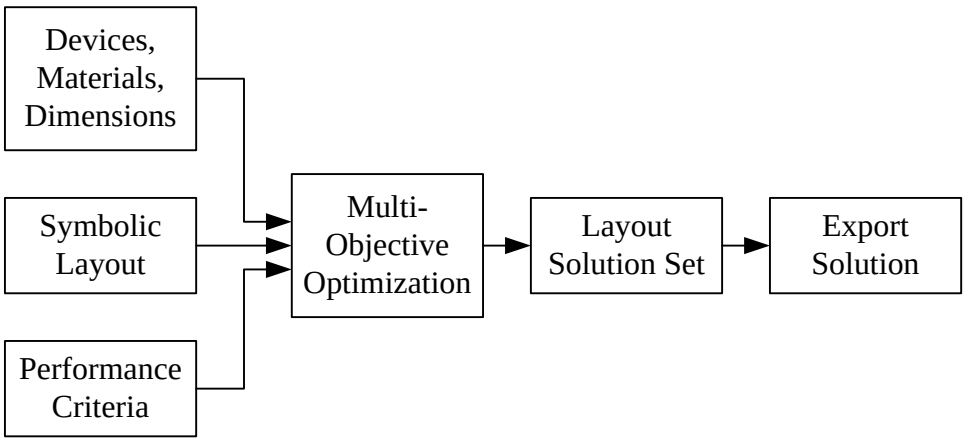


Fig. 1.5: Overall software flowchart.

PowerSynth is composed of three main user interfaces. The first is the Technology Library Editor (TLE) which provides an interface for editing the technology library. The second is the Project Builder. It is used to create MCPM design projects and input the information prior to executing the multi-objective optimization process. The third is the Solution Browser which contains an interactive plot used to select solutions derived from the optimization process. The solutions are plotted against the chosen performance criteria and may be filtered based on desired ranges of criteria e.g. temperature, loop inductance, etc.

1.7 Thesis Outline

Chapters 2 through 5 of this thesis describe the main independent components which form the basis of PowerSynth. Chapter 2 summarizes the thermal modeling approach and presents the automatic device characterization process. Chapter 3 summarizes the electrical parasitic models and elaborates the parasitic extraction and measurement system. An introduction to multi-objective optimization and the specific algorithm used in PowerSynth is given in Chapter 4. All aspects of the symbolic layout system from the loading process of symbolic layouts to the generation of trace geometry and component placement is described in Chapter 5. Chapter 5 also discusses the process design rules and design rule checking for MCPMs.

Chapters 6 and 7 present the integration of these main components into a single software tool. Chapter 6 describes the organization of PowerSynth in terms of data structures and the data flow of the tool from input to output. The graphical user interfaces which guide a user through technology library editing, project setup, and solution selection are discussed in Chapter 7.

The design of a half-bridge module using the P-Cell N-Cell layout technique with PowerSynth is presented in Chapter 8. Human designs are compared with design results using PowerSynth. The thesis is concluded in Chapter 9 with a summary of the work, conclusions gained from the project, and recommendations for future work.

Chapter 2 Thermal Modeling and Automatic Device Characterization

2.1 Introduction

The thermal modeling approach developed for the power module layout synthesis tool is presented in [7] and [8] in further detail, but it is covered briefly in the next section before the automatic characterization process is explained in detail. This fast thermal model is sufficiently accurate to be effective in an optimization framework and yet typically executes four orders of magnitude faster than FEM models. The fast thermal modeling method requires an initial characterization step where a finite element method (FEM) based simulation is conducted in order to find temperature and heat flux distributions which emanate from a die given the materials and dimensions for the baseplate and substrate of a module. These distributions allow for prediction of die temperatures in a module with modifications to trace layout, die positioning, and die quantities. The automatic characterization process of a fast, low order thermal model presented in this chapter describes how an FEM model of a module is constructed, meshed, and solved automatically to gain temperature and heat flux distribution information without any manual labor from the user. This reduces errors in the characterization process and greatly reduces time and effort of introducing new devices and new module configurations into the tool.

2.2 Thermal Modeling Algorithm

The thermal modeling algorithm is designed around the general structure of an MCPM which can be seen in Fig. 2.1. Some power semiconductor die reside on the top layer of the MCPM. These die produce heat energy, due to ohmic and switching losses, which flows downward through the module where it is released into the ambient environment at the bottom

surface of the baseplate. An effective convection coefficient is used to model the process by which heat flows from the baseplate bottom surface into the ambient environment. The baseplate may in actuality be attached to an air or water cooled heat sink, but the effective convection coefficient allows for modeling of these different cooling systems without complex simulation techniques. The rest of the surfaces of the module are assumed to be perfectly isolated from the surrounding environment such that all heat energy flows by conduction through the solid materials which comprise the module [7], [8].

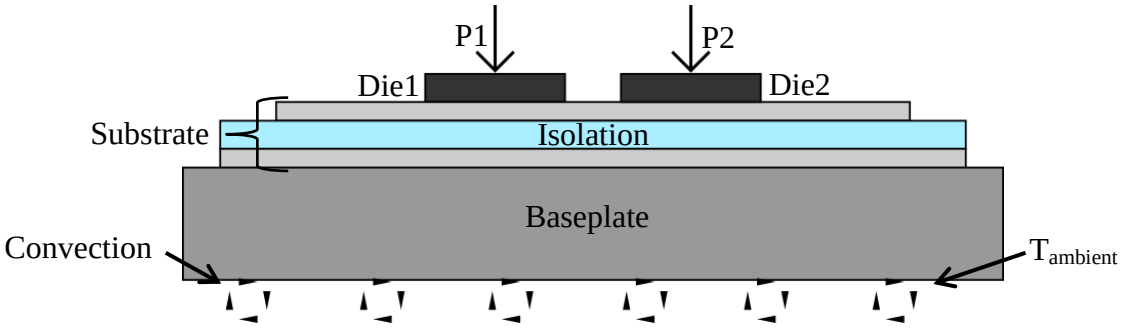


Fig. 2.1: Cross section of MCPM showing thermal boundary conditions.

A lumped element heat transfer network topology, shown in Fig. 2.2, is developed based upon the previously described heat flow problem through an MCPM. A lumped element heat transfer network models heat flow in a system analogously to electric currents in an electric circuit. The materials which comprise the different layers of an MCPM each have some finite thermal conductivity which corresponds to the thermal resistances shown in Fig 2.2. As heat energy flows through the thermal resistances in the network, represented by P1 and P2 in Fig. 2.2, temperature differences or voltages by analogy are formed across each thermally resistive layer [7], [8].

The temperature value at each node represents the average temperature across the corresponding surface of that layer. For example, the top surface of Die1 in Fig. 2.1 is represented by the node which connects the heat source or current source in Fig. 2.2 to the left side of thermal resistance, R_{die1}. The total thermal resistance of all of the layers from the bottom surface of the baseplate to the top surface of the substrate is formed into a lumped thermal resistance, R_{sub} [7], [8].

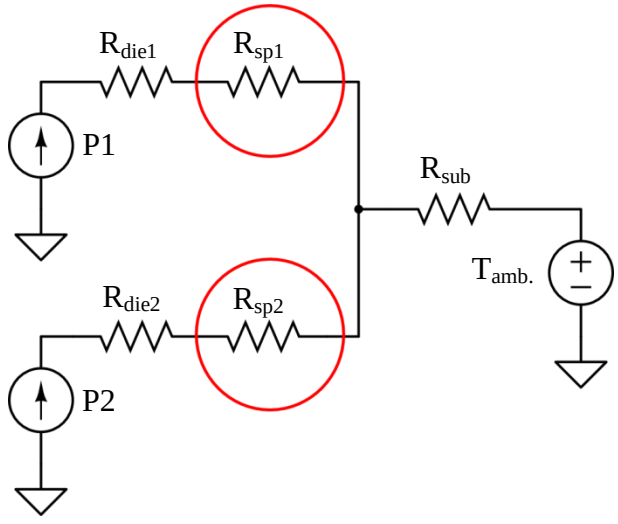


Fig. 2.2: Lumped element heat transfer network representation of MCPM.

The resistors circled in red in Fig. 2.2 represent a certain type of thermal resistance called thermal spread resistance. This type of resistance does not correspond to a physical layer in the module, but to an effect which occurs when heat flows from a layer with smaller cross-sectional area to larger cross-sectional area. The heat flux in one layer is at a higher density than in the succeeding layer. Thus, there is an interstitial portion of space where the heat flux is spreading out to match the lower density of flux. This spreading of heat flux is approximated by the spread resistances, R_{sp1} and R_{sp2}, in Fig. 2.2 [7], [8].

As the layout of a module changes, the bulk of the thermal resistance change occurs in the spread resistances. As long as the overall size and material properties of a module do not change, the change in spread resistance between the bottom surface of the die and the top surface of the trace can be accurately approximated with changes in trace layout and die positioning. The spread resistance is approximated using temperature and heat flux distributions characterized for each die used in an MCPM. If a number of die are identical (the same dimensions and material), the same characterization can be used for the group [7], [8].

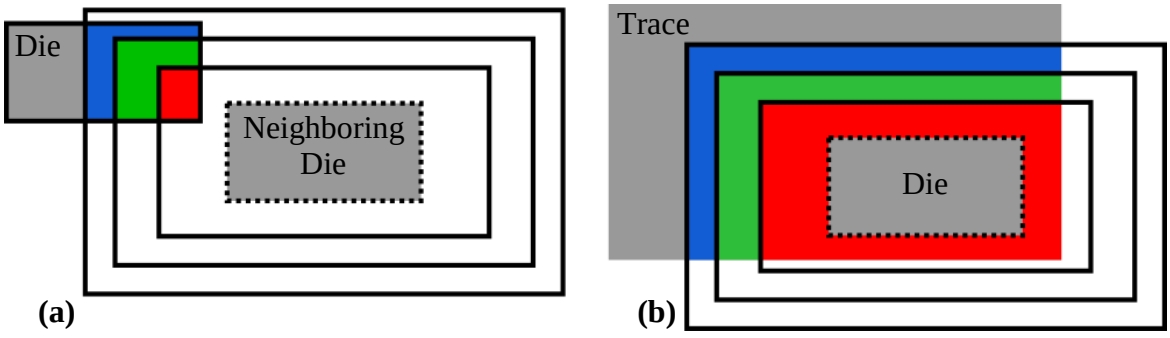


Fig. 2.3: Thermal model rectangular contours. (a): Temperature contours intersecting with neighboring die. (b): Heat flux contours intersecting with trace.

The thermal modeling algorithm breaks the spreading resistance into two major components. The first is a thermal coupling component which is calculated for a given die using

the superposition of neighboring die's temperature distributions onto the die's footprint, shown in Fig. 2.3(a). The temperature distribution of each neighboring die is represented as a set of rectangular contours (Fig. 2.4). The rectangular representation of the distribution allows for fast integration of the neighboring temperature distributions in the region of a die's footprint [7], [8].

The second is an edge effect component which is calculated using the heat flux distribution for a given die, shown in Fig. 2.3(b). Again, the heat flux distribution is represented by a set of rectangular contours. Except this time, the contours are intersected with the trace in which the die is mounted on. As the die approaches closer or farther away from the edge of the trace, more or less of the contours intersect with the trace. When more contours intersect with the trace, more effective cross-sectional area exists for heat energy to flow through thus decreasing the spread resistance [7], [8].

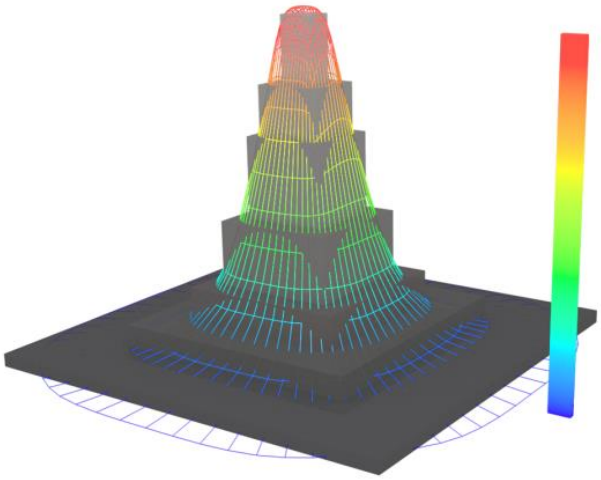


Fig. 2.4: Temperature distribution overlaid with rectangular contour approximation.

The temperature and heat flux distributions used in the thermal model discussed above are acquired through an FEM-based model of an MCPM. Every type of heat producing device

used in an MCPM such as diodes or MOSFETS, needs to undergo a thermal characterization. A separate FEM model is built and solved for each type of device and the temperature and heat flux distributions for each are fitted and saved. Further information about the rectangular contour fitting process is described in [7], [8].

2.3 Automatic Thermal Characterization

2.3.1 FEM Tool Background

In order to conduct a finite element analysis (FEA) on a model, a meshing system and FEM solver are needed. The meshing system breaks a geometric model of a system into a finite and discrete set of elements or finite elements. The elements are tied together based on their shared boundaries with each other by equations which approximate a particular partial differential equation (PDE). As a geometric model is further subdivided into smaller elements, a better approximation to a true solution of the PDE is achieved, yet simultaneously this increases computational time and cost. The heat equation is the PDE used for modeling the thermal conduction processes in this particular section [10].

The solver portion of the FEA process takes the meshed elements of the geometric structure and based on the boundary conditions placed on the system and chosen PDE outputs a solution to the problem. At a more detailed level, the elements are tied together by equations and thus form a large system of equations which are represented by a matrix and input vector. The input vector for a heat equation problem would be the heat flux entering the system at the top of a device. An output vector representing some value, such as temperature, at each node in the mesh is found by the solver. The solver may be able to use a number of techniques to solve the large system of equations depending on the type of problem at hand [10].

2.3.2 FEM Tool Selection

ANSYS and SolidWorks are two different tools used to build thermal simulations used for manual characterization. Each of these two tools combines geometric design, meshing, and FEM solving into one software package [11], [12]. Either tool is sufficient for conducting the simulations needed to gather thermal characterization data for modules, but in order to automate the design process some additional aspects are considered such as automation capability, operating system portability, data extraction, simulation accuracy, and software licensing cost.

In order to address some of these issues, open-source alternatives for meshing and solving finite element problems were explored. The open-source meshing tool, Gmsh, allows for automated mesh generation from a geometry description file [13]. The open-source FEM solver Elmer can solve a desired PDE on a mesh generated by Gmsh [14]. Using these two tools in conjunction with each other allows for a complete automated characterization process starting from design geometry input to thermal data output.

Both of these open-source tools, as well as ANSYS, allow for great portability as they are available on Windows and Linux based operating systems. SolidWorks on the other hand is only able to execute in a Windows environment. Both SolidWorks and ANSYS have automation capabilities, so all of the tools considered cover this criterion. Data extraction capability is also covered by all of the above tools. In terms of simulation accuracy, Gmsh and Elmer provide accurate results as compared to SolidWorks for similar thermal problems. A comparison of the two is shown in Section 2.3.4. Since ANSYS and SolidWorks are both commercial tools, a software license needs to be obtained in order to operate them. The open-source license used by Gmsh and Elmer allow for wide distribution of the tools for free. Under these criteria, Gmsh and

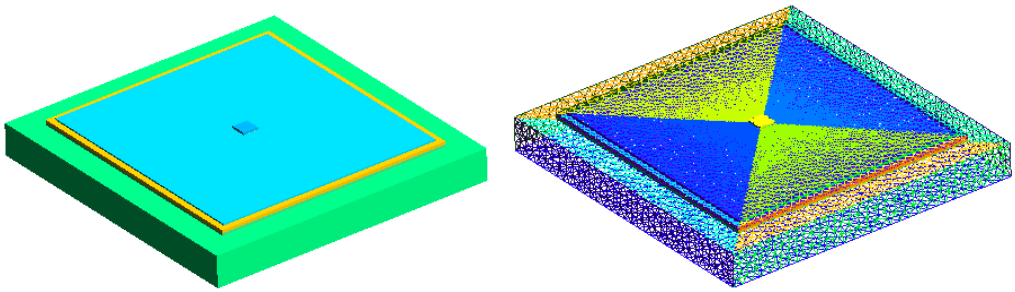
Elmer were selected for automation of the thermal characterization process. The following sections detail the meshing and finite element analysis setup used in PowerSynth.

2.3.3 Meshing with Gmsh

The Gmsh meshing program outputs a mesh for some input geometry. Gmsh takes a file format with extension ".geo" that contains instructions on how to build a specific geometry in either 2D or 3D space. Once the ".geo" file is loaded and constructed, the geometry can be meshed.

Geometry in a ".geo" file is described by a special scripting language interpreted by Gmsh, and is built up from a set of points, lines, surfaces, and volumes. Each point, line, surface, etc. is created as a programming object in the scripting language and can be manipulated as such. The first primitive components used to describe the model's geometry are collections of points which can be used to form lines. Next, collections of lines are used to form surfaces and collections of surfaces to form volumes.

A rendering of a loaded geometry file is shown in Fig. 2.5(a). The module structure for the thermal characterization is a stack of rectangular volumes (boxes) in contact with each other. Since the ".geo" file is a script, it is written to take a list of box dimensions as input each with a width, length, and thickness. Each box is assumed to be centered at the origin. The first three box dimensions in the list represent the lowest box on the z-axis (the baseplate), and the last dimensions represent the highest box (the die under characterization). The script takes the list of box dimensions and generates the points, lines, surfaces, and volumes associated with each box at the proper locations in space. The ".geo" file or script can be found in Appendix A.



(a) **(b)**

Fig. 2.5: Module characterization setup with die in center. (a): Surface rendering of module. (b): Module model with meshing.

After the model geometry is constructed in Gmsh by loading the ".geo" file, the model is meshed. The granularity of the mesh is selected by applying a characteristic length to each point in the model, e.g. the box layer corners. Gmsh will attempt to match the edge lengths of the tetrahedron finite elements to the characteristic length of each point and linearly interpolate the lengths between points accordingly. The frontal meshing algorithm is selected in Gmsh as the meshing algorithm which produces tetrahedral elements. Gmsh ensures nodes on surface boundaries match and that the entire model forms a coherent solid mesh [13]. The surfaces of these tetrahedral elements can be seen in Fig. 2.5(b) and Fig. 2.6.

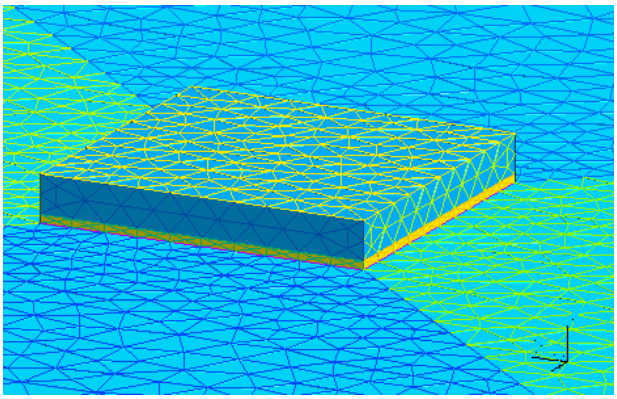


Fig. 2.6: Close-up of meshed die and die attach layers.

The characteristic length used in the meshing of relatively thin layers needs to be considered. The solder or die attach layers are usually thin with respect to the surrounding die and metal layers. If the length chosen is too small, the meshing algorithm may take long amounts of time in order to create a fine enough mesh for the thin layer or may even fail. If the length is too large the meshing algorithm may generate incorrect or badly sized elements for the thin layer. Using the same characteristic length as the die mesh is found to work in many situations. This may be due to easing the node matching between the die and solder layer. The interaction between these two layers can be seen in Fig. 2.6.

The entire process described above can be executed via command line by Gmsh, thus no graphical user interface interaction needs to take place. Gmsh is called with arguments which point it to the correct ".geo" file and mesh algorithm selected. Gmsh reports back as to whether the meshing process was successful or not.

2.3.4 Finite Element Analysis with Elmer

After the mesh generation process, the FEA problem needs to be set up and solved. This is achieved by writing a ".sif" file which is passed to Elmer, the FEM solver. The materials of the model need to be described in terms of density and heat conductivity in order to solve the heat equation. The boundary conditions of the model also need to be setup.

The different material properties of each layer in the model are applied and identified by volume numbers given to each layer in the original ".geo" file used by Gmsh. Gmsh retains these identification numbers in the output mesh file. Collections of points, lines, or surfaces can also be identified in the same way. Each layer is given a unique volume number such that the correct

material properties can be properly attributed to each layer in the ".sif" file. An example ".sif" file is shown in Appendix B.

The boundary conditions are assigned using a similar method, except surface identification numbers are used instead. The top surface of the die is given a numeric identifier, and a heat flux value is assigned to the surface corresponding to the total heat flow power dissipated from the die for characterization. The next boundary condition, a convection process, is applied to the bottom of the baseplate through the same method. All of the other outer surfaces default to a perfectly isolated state.

After the problem has been defined, the system needs to be solved. The solutions to a finite element system are never perfect. There is some tolerance of error allowable in the solutions, similar to a SPICE solver. If the convergence tolerance is set too large, the amount of error in the solution set may be too large. If the convergence tolerance is set too low, the solver may never converge to a low enough error measure. A convergence tolerance of 10^{-7} is found to allow the Elmer solver to converge on a solution quickly and also maintain a reliable amount of fidelity in the solution data (the temperature at each mesh point).

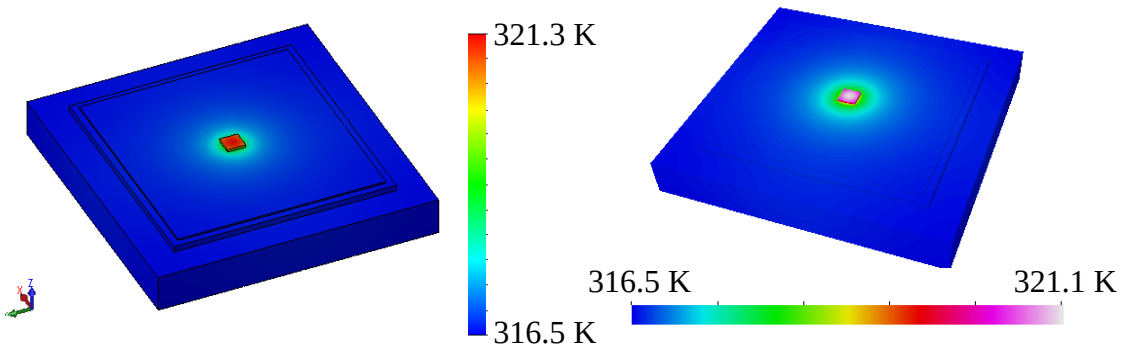


Fig. 2.7: Steady State Thermal Simulation Results. (a): Result from SolidWorks. (b): Result from Elmer.

A comparison between a model built, meshed, and solved in SolidWorks and Elmer can be seen in Fig. 2.7. The overall temperature distributions in both models are quite similar. Table 2.1 summarizes the minimum, maximum, and average temperature of the two different model solutions. The Elmer minimum temperature result has 0% error with 5 significant digits of precision, and the maximum and average temperatures each have less than 0.1% error, as compared to SolidWorks results. Therefore, Elmer is able to provide high enough quality thermal solutions in order for use in the thermal characterization process.

Table 2.1: SolidWorks vs. Elmer FEA results.

	Min. Temp. (K)	**Max. Temp. (K)**	**Avg. Temp. (K)**
SolidWorks	316.59	319.33	316.8
Elmer	316.59	319.28	316.7

Elmer outputs the solution data in an ASCII text file. The file contains a list of the (x,y,z) coordinates of every node in the problem mesh and a unique integer ID for each node. Each line of node data in the file is delimited by a newline character. A list of temperatures at every node is listed below the coordinates and node IDs in the same order. A separate list of heat flux components flowing in the x, y, and z directions is also contained in the file in a similar manner to the temperatures. The file is parsed for nodes which reside at the same z level as the top of the isolation layer. Once the correct nodes are determined, the temperature and heat flux data is extracted for each distribution. Next, this data set is passed through the fitting portion of the characterization process detailed in previous work [7], [8]. This allows the entire characterization process to be completed without human interaction.

Chapter 3 Electrical Parasitic Modeling and Extraction Algorithm

3.1 Introduction

The previous work conducted on electrical parasitic modeling, described in Gong's thesis, primarily focuses on the development of physical models that gauge parasitic resistance, inductance, and capacitance rather than the extraction algorithm. The goal of the parasitic extraction algorithm is to produce a circuit representation of a power module in which electrical parasitics are included. This is achieved by breaking a layout design into basic enough elements such that the physical models can be used to evaluate the individual elements' electrical parasitic properties. Once each element's parasitic values are known, they are written into a lumped element circuit representation of the module. Some initial work in parasitic extraction is covered in Gong's thesis, but only a specific case is considered. This work served as a basis for further development of the extraction algorithm. This chapter will reiterate some of the previous work on the physical models and their parameters and describe further developments in the parasitic extraction algorithm [8].

3.2 Parasitic Resistance and Inductance Modeling

A micro-strip transmission line structure is used to model the parasitic resistance and inductance in the traces of MCPMs. Fig. 3.1 illustrates the similarities in structure between the two systems. The micro-strip structure consists of an infinite ground plane, a dielectric layer, and a top metal layer. The MCPM structure consists of a finite ground plane (baseplate), a dielectric layer (isolation), and a top metal layer (trace). The primary difference between these two structures is the finite versus infinite ground plane. The infinite ground plane approximation of

the micro-strip line structure models the parasitic resistance within 10% accuracy of a FEM model of the system, but does not model the parasitic inductance with as much accuracy. The ground plane has a much more dominant effect on the inductance of the traces in the system, thus having a finite dimensioned ground plane affects the quality of the inductance model. In order to improve accuracy, an average between the inductance of an isolated bar and the micro-strip line is taken. This averaged model improves the error for the inductance to around 10% for individual trace lines [8].

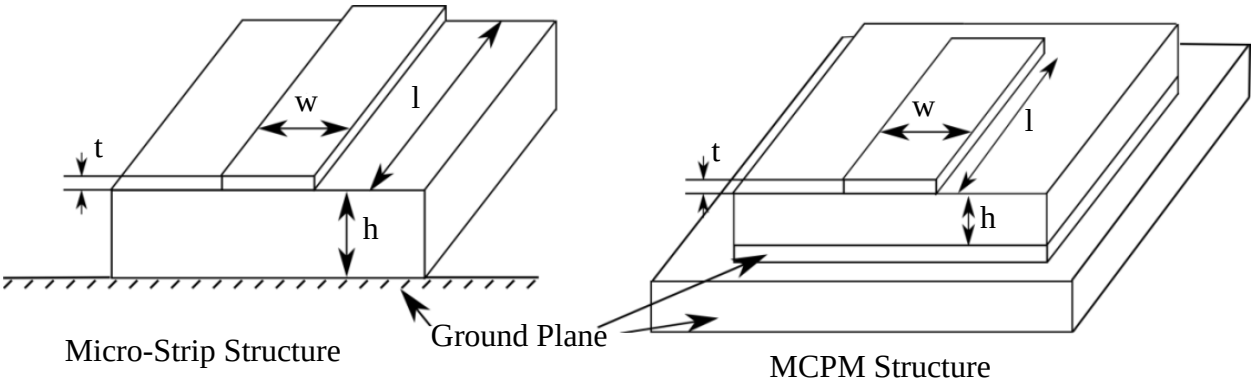


Fig. 3.1: Comparison of micro-strip transmission line structure and MCPM structure [8].

The main benefit to modeling the inductance and resistance of the traces with a micro-strip transmission line model is that a closed-form equation is used to find the values based on simple geometric and material parameters. If an MCPM layout is broken into strips of rectangular trace, the micro-strip equations can be easily applied to this system and at very low computational cost to estimate the parasitics of the total layout. In terms of computational cost, the algorithm was found to run around 1 million times faster than a full FEM analysis.

Another contribution to parasitic inductance and resistance in a layout comes from the bondwires which connect from device to trace or trace to trace. Closed-form equations which

include frequency dependent effects such as the skin-effect were researched and found to model the inductance and resistance of bondwires effectively. Although further research is required in order to consider the effects of mutual inductance between the wires. Documentation regarding the verification and equations of the previously described models can be found in [8].

3.3 Parasitic Capacitance Modeling

A similar closed-form equation based approach is used for modeling the parasitic capacitance of the traces in MCPMs. The primary capacitance for each trace exists between the second metal layer and the top metal layer of an MCPM, as seen in Fig. 3.2. The coupling capacitance spanning from trace to trace is typically several orders of magnitude smaller than the aforementioned substrate capacitance, thus is not included in the modeling effort. Coupling capacitance between traces is generally quite small due to the small vertical sides of the trace with respect to the distances in which trace is separated. Coupling capacitance plays a larger role in integrated circuit layouts because the interconnect aspect ratio is closer to 1.0 and horizontal distance between interconnect is relatively closer. Mutual inductance between traces plays a far greater role in unwanted coupling inside a power module. Research effort spent on modeling mutual inductive coupling would be far more effective in exploring coupling issues [8].

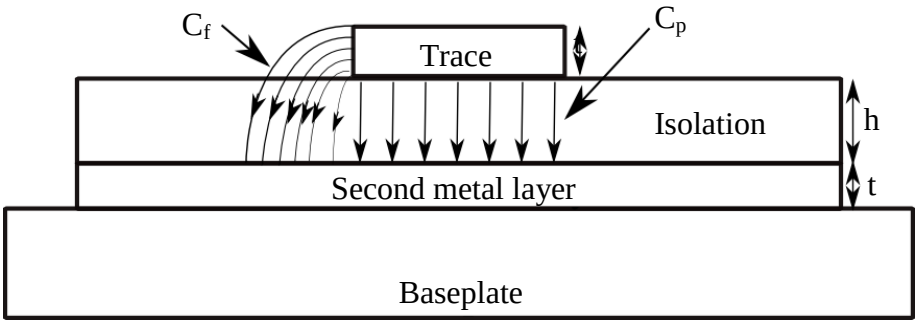


Fig. 3.2: Parallel plate and fringing field lines which make up parasitic capacitance [8].

The substrate capacitance, measured from the trace layer to second metal layer as shown in Fig. 3.2, is estimated by summing together the capacitance from the parallel plate capacitance equation and a fringe capacitance equation. The fringe capacitance equation approximates the amount of capacitance due to field lines extending from the sides of a piece of trace to the substrate. The model increases in error as the distance between the trace and substrate is increased, due to the complex effects of the fringing field lines around the perimeter of the trace. The fringing field lines pass through both the passivation coating and the isolation dielectric material which affects the fringe capacitance value. Since the passivation coating is typically quite thin as compared to the thickness of the isolation layer, the permittivity for the passivation material is left at a value of air or vacuum. The summed model of simple parallel plate capacitance and fringe capacitance yields less than 7% error for typical power module parameters and reaches a maximum of 26% error as isolation thickness, h, is increased to a maximum of 0.8 mm in the verification study presented in [8]. A typical isolation thickness for a high-voltage (1200 V) MCPM substrate is around 0.64 mm or less. This is well within bounds for accurate substrate capacitance estimation.

3.4 Parasitic Extraction Algorithm

A graph is used as the primary data structure to hold the extracted parasitic information. A graph is composed of two basic elements known as nodes or vertices and edges. The circuit schematics electrical engineers use every day are also simple graphs with nodes and edges. In circuit schematics, the nodes are known as nodes and the edges as branches. Passive circuit elements such as resistors, capacitors, and inductors span two nodes and thus constitute a branch. An example parasitic graph is shown in Fig. 3.3, where each edge holds inductance, resistance,

and substrate capacitance values between layout nodes. This graph is not a circuit schematic, although it bears a close resemblance. The nodes in Fig. 3.3 represent either connection points between traces labeled in green, lead connections labeled in blue, or device connections labeled in red. A circuit schematic in the form of a SPICE net-list can be constructed from this parasitic graph, and the process is detailed in Tucker's Honor's thesis [15].

The graph data structure is implemented using a Python graph library called NetworkX [16], [17]. A NetworkX graph is constructed by adding nodes and edges to a graph programming object. Multiple attributes can be attached to any node or edge in the graph. An attribute consists of a key-value pair. The key is usually a string which gives the name of an attribute, and the value carries the value of the attribute. For example, the edges in the parasitic graph each have at least three attributes with key names "res", "ind", and "cap" representing resistance, inductance, and capacitance, respectively. The particular values for each key are floating point numbers representing the amount of resistance, inductance, or capacitance in the edge element. The value of attributes associated with nodes and edges need not be simply numbers. The nodes which connect to devices or leads have attributes whose values reference a programming object which holds various parameters and methods for that device or lead.

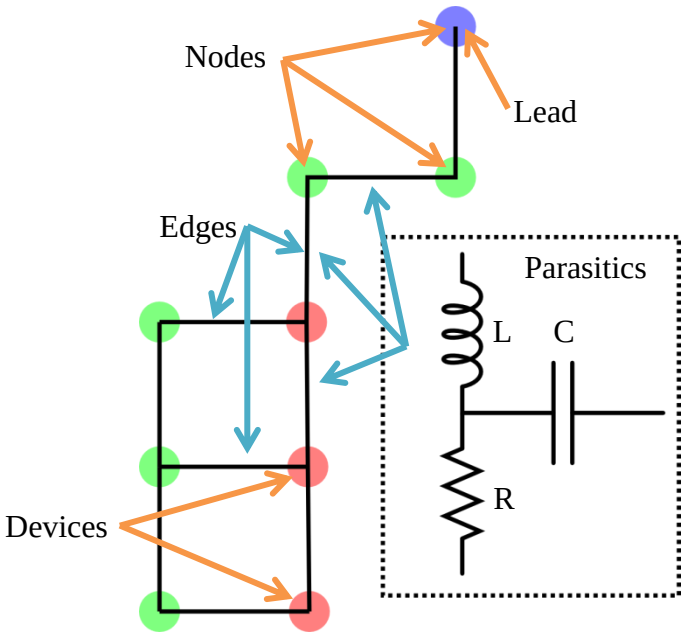


Fig. 3.3: Graph based representation of parasitic network.

The layout of the power modules in this thesis are described by a set of rectangular traces. The individual trace rectangles can be seen in green in Fig. 3.4, where each is surrounded by a light black border line. The parasitic graph is created based on this set of rectangles. Each trace rectangle is oriented in either a horizontal or vertical way except for a special case of trace rectangle called a super-trace. A super-trace has neither vertical nor horizontal orientation in particular and is dealt with on both axes. The terms horizontal and vertical in this context refer to the x-axis and y-axis, respectively, and not vertical in the z-axis sense.

Since each trace rectangle has some orientation associated with it, a set of points can be formed along the length of each trace rectangle. This set of points or nodes, called spine nodes, run along the length of a trace and reside in the middle of the width of a trace. The spine nodes are placed based on whether there is a connected trace, device, lead, or bondwire along the

length of the trace. The spine nodes for a vertical trace rectangle are pointed out in Fig. 3.4. The first point from the top is a trace connection node and the next three are device connection nodes.

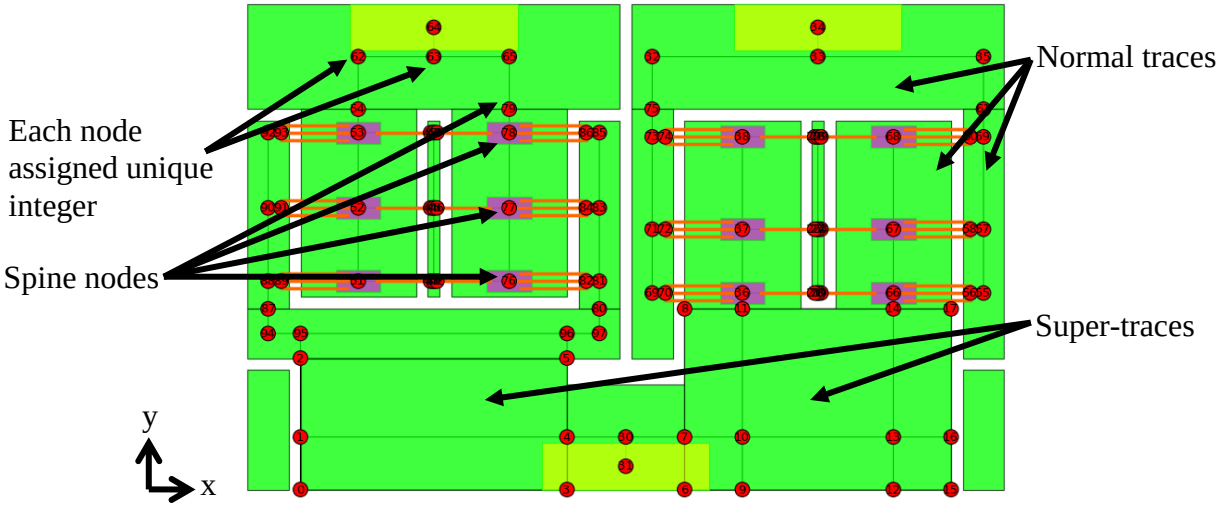


Fig. 3.4: Module layout with overlaid parasitic graph.

The spine nodes are allocated by iterating through each trace rectangle object and identifying if there are leads, devices, bondwire connections, or trace connections made at certain points along the length of the trace. This information is bound to the trace rectangle object through another data structure in the system called a symbolic layout. Symbolic layouts are described in more detail in a forthcoming chapter, but basically contain information about the connectivity of traces and what leads and devices reside on traces in a symbolic form rather than a purely geometric form.

Once the spine nodes are added for a given trace, they are sorted in geometric order from lowest to highest. For example, if a trace is oriented vertically, the spine nodes will vary from some lower y coordinate to some higher y coordinate. After the spine nodes have been sorted, they are iterated over from lowest to highest coordinate and connected together with graph edges ensuring the nodes are connected together in correct geometric progression. Each spine node is

associated with one other node, either connecting to a device, lead, bondwire, or trace connection. Edges connecting these “off spine” nodes are also created, forming interconnections between traces and other objects in the network. Sometimes the connected object resides directly on the spine of the trace, such as the devices positioned in the middle of the traces in Fig. 3.4, thus an “off spine” connection node and edge is not needed.

The super-trace nodes are allocated differently. The geometric representation of a super-trace is still a rectangle. The super-trace rectangle is able to span multiple normal trace rectangles. There exist two super-trace rectangles in Fig. 3.4, in which they span two normal vertical trace rectangles above them. The super-trace rectangle on the right in Fig. 3.4 spans two horizontal trace rectangles also. Since a super-trace is able to span multiple horizontal and vertical trace rectangles, this means multiple connections can be made to the trace on both axes. Instead of a single spine of nodes, a mesh of nodes and edges is needed to facilitate the multiple connections which can be made on both axes.

The mesh is generated by considering the horizontal and vertical axes separately and finding trace connections made on either axis. Next, a matrix type mesh is created based on the number of trace connections made on the vertical and horizontal axes. On each axis, initially two points are added regardless of any trace connections on them. Next, a point is added for each trace connection on each axis. The total number of nodes in the matrix mesh will be the number of points on each axis multiplied by each other. For example, the right-hand super-trace in Fig. 3.4 has two vertical traces connected to the top of it and one horizontal trace connected to the left-hand side of it. This yields 4 points on the horizontal axis and 3 points on the vertical axis.

Thus, a total of 12 nodes are seen in the body of the mesh of the super-trace. The resulting mesh is a non-uniform grid.

As the nodes for each trace rectangle are being created and connected together with edges, the parasitic models are also evaluated based on the particular dimensions of each trace and the distance between two nodes on the spine of each trace. Each of these pieces is referred to as a trace segment. These trace segments are rectangular in shape which allow the micro-strip transmission line equations and parallel plate capacitance equations to be easily evaluated for each. Fig. 3.5 illustrates trace segmentations in one half of a layout on the right side of the image, and the associated lumped element network of inductances on the left. This figure is to demonstrate that there is some parasitic inductance, resistance, and capacitance associated with each trace segment. The trace segments which run along the spine of a trace take on the width of that trace. In the case of trace segments which do not reside on a trace spine, such as a segment connecting to a lead or bondwire connection, these take on the width of the connected lead or total width of the connected bondwires.

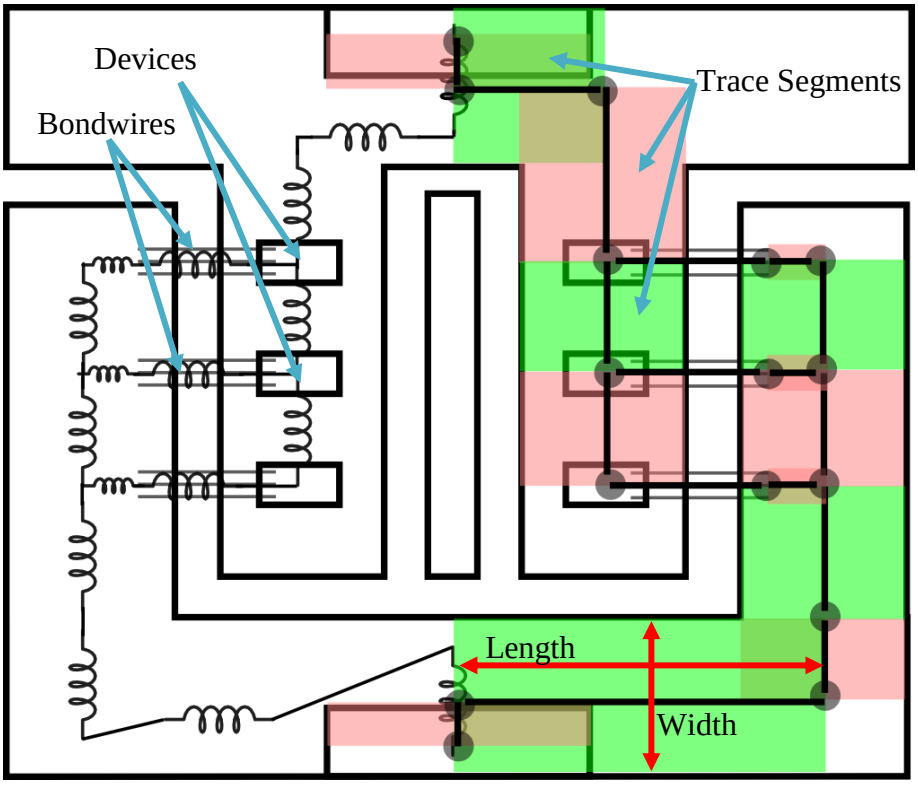


Fig. 3.5: Parasitic extraction lumped element network and trace segmentation.

The super-trace edges also need a segment width and length in order to allocate an accurate parasitic value for each edge element. Fig. 3.6 illustrates the segmentation concept for a 4 column and 3 row super-trace meshing. Fig. 3.6(a) and 3.6(b) show the vertical (column) and horizontal (row) segmentation cases, respectively. The red and green rectangles represent the geometric size of each segment and the black lines with double arrowheads represent the corresponding graph edges. The segment length for each edge (vertical or horizontal) is simply the length of the edge itself or the distance between the two nodes which make up the edge.

The width segmentation process is also dealt with in terms of rows and columns which are denoted by (X_1, X_2 … X_N) where N is the number of rows or columns (Fig. 3.6). Each graph edge receives some segment width (C_1, C_2 … C_N) which is shared with each edge in the row or

column. The graph edges on the super-trace boundaries have widths C_1 and C_N which take on a width equal to half the horizontal or vertical distance to their nearest neighboring edge represented by B_1 or B_{N-1}. The symbols A_n in Fig. 3.6 represent the distance between column or row X_n and X_{n+1}, and B_n represents half this distance. The internal columns or rows (X_2 … X_{N-1}) take on a width of $B_{n-1}+B_n$ where n is the current column or row. This width segmentation process ensures that the total width of the segments (C_1 … C_N) is equal to the width of the super-trace in the vertical and horizontal directions.

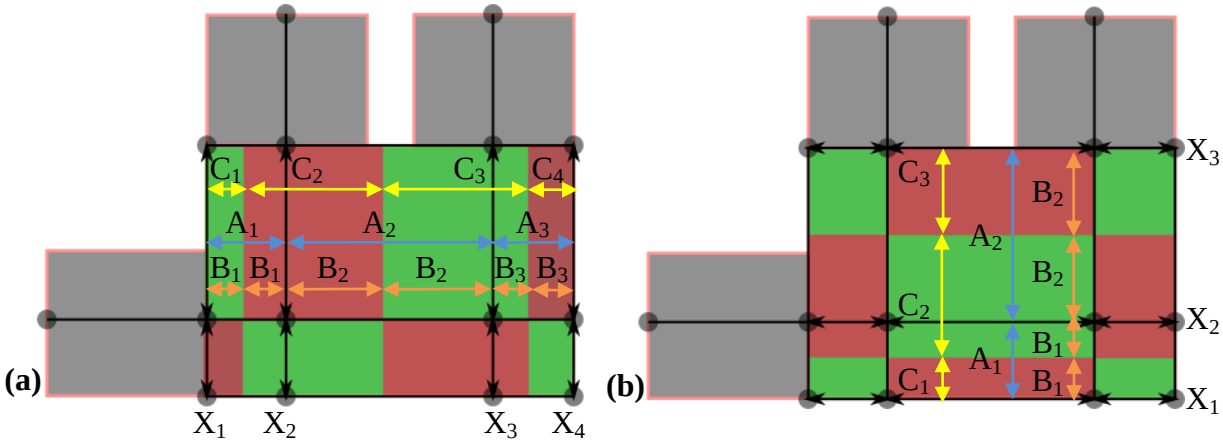


Fig. 3.6: Super-trace segmentation with segment widths marked with yellow lines. (a): Vertical segments. (b): Horizontal segments.

The micro-strip transmission line equations used for calculating the inductance of each segment in the extracted layout only yield accurate results for segments whose aspect ratios are less than 1.0 [8]. The aspect ratio, defined as width divided by length, of segments in layouts is frequently greater than 1.0. In order to help approximate the inductance of segments more effectively, the maximum aspect ratio of segments is constrained to 1.0 by automatically reducing the segment width to be equal to the segment length in these cases. The inspiration behind this solution is that the current in wide traces is not usually uniformly distributed

throughout the trace anyway. A reduction in the effective width of the trace generally helps improve the inductance estimation for most layouts.

3.5 Parasitic Network Measurement

It is useful to be able to measure the effective resistance or inductance between two different nodes in the extracted parasitic network, such as measuring the loop inductance in an H-bridge or the gate loop inductance between the gate and source leads of switching positions. The extracted parasitic network takes the form of a graph and can be converted into a Laplacian matrix from this graph data structure. The Laplacian matrix of a graph is similar to the admittance matrix of a circuit used in SPICE simulators [18], [19]. The construction of a Laplacian matrix allows fast measurements of resistance or inductance between nodes. Fig. 3.7 shows a simple parasitic graph with each node given a unique integer and its corresponding Laplacian matrix to the right. The node integers index the rows and columns of the Laplacian matrix.

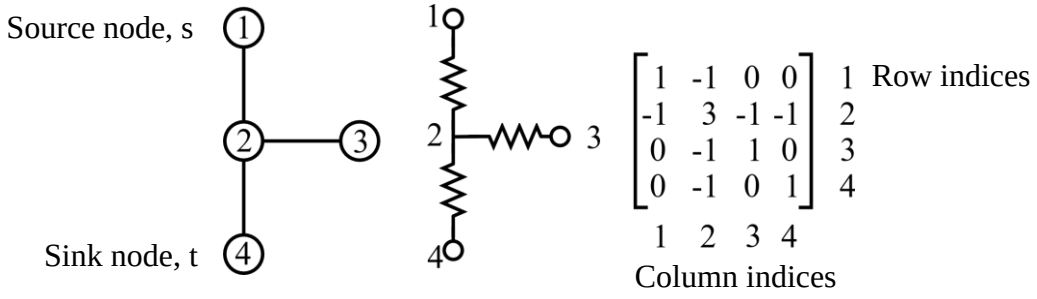


Fig. 3.7: Graph and Laplacian matrix with all edges weighted to 1.0 (1.0 Ω).

The Laplacian matrix is constructed either by an element stamping method, nodal analysis, or some other means [18]. Fortunately, there exists a function in the NetworkX graph library which can produce a Laplacian matrix from a graph. Although, a weight value for each

edge in the graph needs to be specified. If the goal is to measure resistance between two nodes in the graph, the desired weight value on each edge would be the conductance or inverse of resistance. If measuring inductance, each edge weight will take on the inverse of the inductance value stored in each edge element. The key-value attribute system described earlier allows the specification of these weighting values for the NetworkX Laplacian matrix function. As long as a key-value pair for conductance and inverse inductance exists for each edge in the graph, a corresponding Laplacian matrix for the system can be constructed for each type.

The effective resistance is measured between two nodes by applying a unitary flow between the two nodes. If a unitary current is made to flow between the two nodes, a potential difference equal to the effective resistance will appear across the nodes via Ohm's law: $V=R$ if $I=1$. The effective inductance can also be found in this way, because inductive elements follow the same mathematical rules as resistive elements when placed in series or parallel configurations. The unitary current applied between the nodes in the inductive case is not a real current, but simply a mathematical tool to aid in finding the effective inductance between the two nodes.

The effective resistance can be found using the Laplacian matrix derived from a parasitic graph via the following linear algebraic equation:

$$R_{eff}^{s,t}=\boldsymbol{x}_{s,t}^{T}\boldsymbol{L}(\boldsymbol{G})^{\dagger}\boldsymbol{x}_{s,t} \tag{3.1}$$

where $\boldsymbol{x}_{s,t}$ is a vector of net electrical flows between a source node s and sink node t, and $\boldsymbol{L}(\boldsymbol{G})$ is the Laplacian matrix of the parasitic graph $\boldsymbol{G}$. The vector of electrical flows $\boldsymbol{x}_{s,t}$ is a vector which has the same number of entries as there are nodes in $\boldsymbol{G}$ and carries the same node indices. $\boldsymbol{x}_{s,t}$ has a 1 in the node index position which represents the source node and a -1 in the position

which represents the sink node. Thus, the net current in the system is conserved. All other entries in $\boldsymbol{x_{s,t}}$ are zero which means no other net currents are flowing between any other nodes in the system. In a SPICE admittance matrix, the row and column entries associated with the ground node are removed e.g. the 4th index in Fig. 3.7. If not removed, a net negative current would be seen flowing in the ground node position of the $\boldsymbol{x_{s,t}}$ flow vector. These rows and columns are not eliminated in this mathematical procedure, because this allows for Eq. (3.1) to be used for effective resistance calculations [19].

$\boldsymbol{L}(\boldsymbol{G})^{\dagger}$ is the Moore-Penrose pseudoinverse of the Laplacian matrix of graph $\boldsymbol{G}$. The pseudoinverse operation is analogous to the inverse of a matrix in a linear algebraic sense. An example of the Moore-Penrose pseudoinverse of a Laplacian matrix is given in Appendix G. It can be used in restricted cases when a matrix is not invertible. The admittance matrix in SPICE solvers is invertible because the ground node rows and columns are removed. In the case of Eq. (3.1), the pseudoinverse operation is acceptable and yields the correct effective resistance or inductance between paths. Although it is possible that a finite effective resistance or inductance value can be calculated between nodes which have no actual path between them (completely disconnected). In order to make sure the two nodes are at least connected, a simple algorithmic check of whether a path exists is performed before the calculation is performed. A proof for Eq. (3.1) and further information regarding effective resistance calculations can be found in [19].

3.6 Parasitic Extraction Results

In order to test the accuracy and effectiveness of the parasitic extraction algorithm, three different layouts (Fig. 3.8) were created and tested with the extraction and modeling methods discussed in the previous sections and compared against finite element method models built with

Ansoft Q3D [20]. Ansoft Q3D is a finite element method based parasitic extraction tool which can accurately estimate resistance, inductance, and capacitance of conductor networks.

The first layout tested, shown in Fig. 3.8(a), only contains two measurement leads, a positive and output. The inductance and resistance between the positive and output lead are measured. The other two layouts, Figs. 3.8(b) and 3.8(c), contain three measurement leads: a positive, negative, and output. In these two cases, the inductance from positive to output and negative to output are measured as well as the resistance between the positive and negative lead. In all three layouts, the total substrate capacitance of all traces either connected to leads or devices (even through bondwires) is measured. This includes the traces connected through gate bondwires. All devices (diodes and switches) in the layouts are shorted through. This is accomplished in Q3D by replacing the devices with copper material of the same dimensions and in the parasitic extraction system by simply analyzing the parasitic graph as is. Table 3.1 summarizes the numerical results of these measurements.

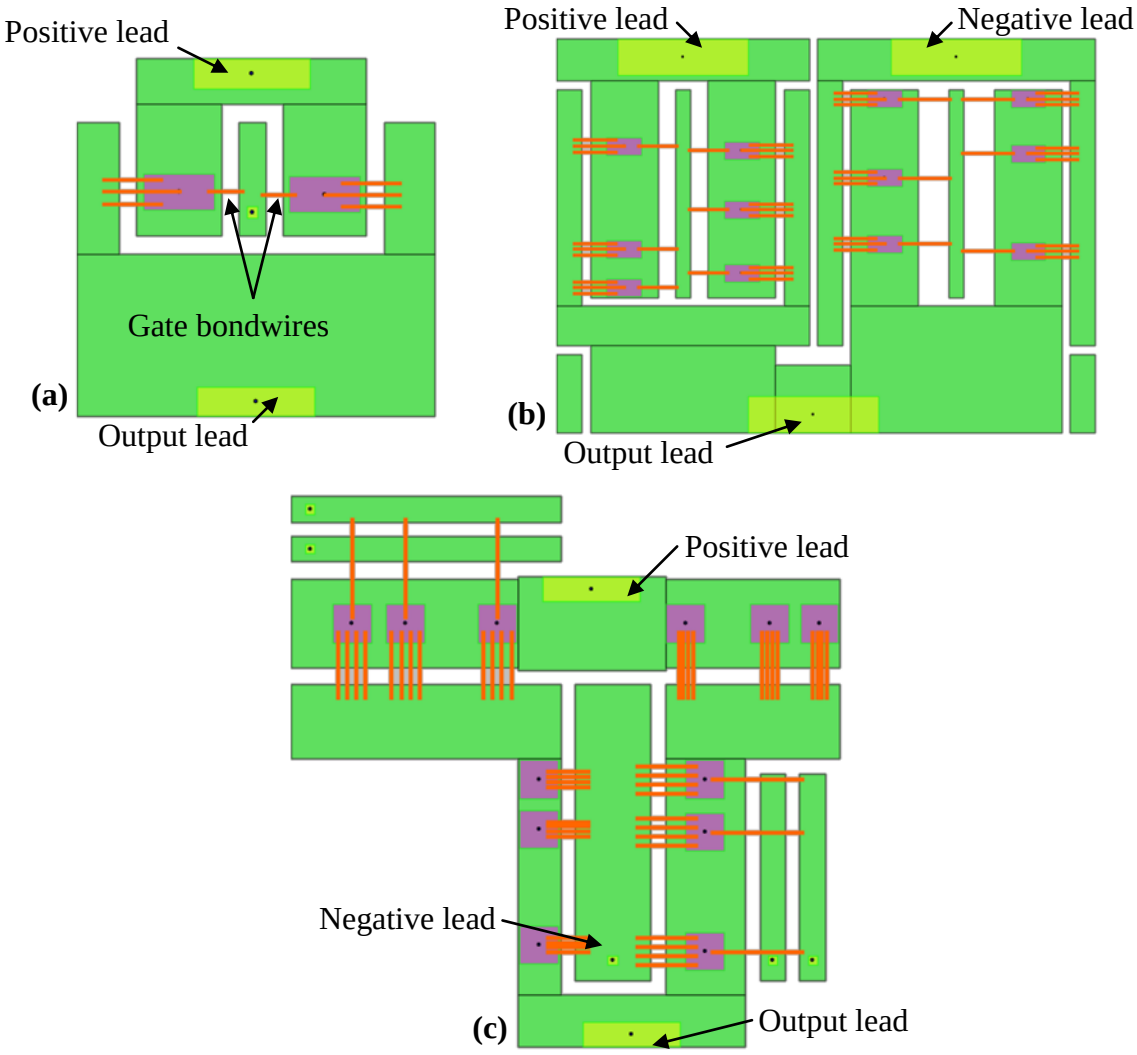


Fig. 3.8: Parasitic extraction test layouts. (a): Simple layout. (b): Complex layout 1. (c): Complex layout 2.

Table 3.1: Parasitic extraction comparison between fast model and Q3D.

	Positive to Output Ind. (nH)	Negative to Output Ind. (nH)	Positive to Negative Res. (mΩ)	Substrate Capacitance (pF)	Evaluation Time (s)
Simple layout (Fast model)	6.476	-	0.6206*	62.98	$2x10^{-3}$
Simple layout (Q3D)	5.155	-	0.3490*	61.78	65
Complex layout 1 (Fast model)	9.973	10.35	0.6270	410.8	$23x10^{-3}$
Complex layout 1 (Q3D)	12.203	12.664	0.98855	418.4	141
Complex layout 2 (Fast model)	7.372	3.090	0.3481	133.6	$10x10^{-3}$
Complex layout 2 (Q3D)	9.253	1.452	0.2914	138.3	124

*Resistance measured from positive lead to output lead.

The inductance extraction results yield less than 2 nH of absolute error, but occasionally has a high relative error rate as in the case of the negative lead to output lead inductance measurement in Complex layout 2. The high error rate in this case, approximately 110%, may stem from shorter inductive paths being evaluated with less accuracy and the compounding factor of smaller inductance values used for calculating the relative error. The lowest error rates for inductance estimation occur in Complex layout 1, which are around 18%. This layout includes longer and thinner sections of trace with lower aspect ratios as compared to Complex

layout 2. The inductive pathways in Complex layout 1 are also more evenly distributed. The negative lead to output lead inductance measure in Complex layout 2 contains a relatively short pathway between the two leads which leads to high segment aspect ratio and again small inductance values. The parasitic inductance model does trend though and is generally able to predict which layout designs yield lower inductive results.

The capacitive extraction model performs the best out of the three models with a highest relative error rate of about 3.5% from Complex layout 2. This is probably because the module fits the parallel plate model quite well. The separation distance between the two metal layers is quite thin with respect to the width and length of the conductors involved yielding good capacitance approximations.

The resistive models performed the worst out of the models with a highest relative error of 77%. This is lower than the highest relative error in the inductance models, but this high error rate occurs for two out of three of the measurements taken. The high error rate may stem from bondwire resistance estimation. The resistance in layouts is dominated by bondwires which is frequently around 10 to 100 times greater than the resistance contributed by the trace. This places the emphasis of accurate resistance modeling on the bondwires.

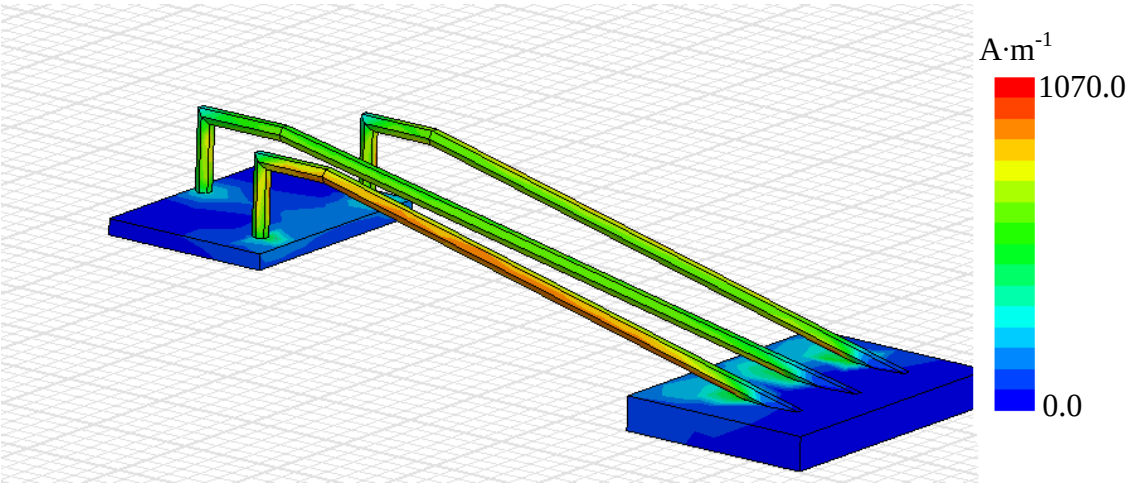


Fig. 3.9: Bondwire experiment showing increased current density in outer wires.

Fig. 3.9 shows the current density on the surface of three bondwires taken from Complex layout 1. There is an increased current density on the outer bondwires leading to increased resistance in the outer bondwires due to the proximity effect [21]. The bondwire model used in the parasitic extraction tool does not consider the proximity effect and thus does not model this resistance as accurately as possible. Further research into the proximity for bondwires would help increase the resistance approximation accuracy.

The parasitic extraction models execute around 10,000 times faster than the Q3D FEM models. The time values given for the parasitic models include the construction of the parasitic graph, inductance, resistance, and capacitance model evaluation, Laplacian based sink to source measurements, and total substrate capacitance summation. The Q3D FEM model execution time depends on mesh fineness: the finer the mesh, the longer the evaluation time. The default settings are used in Q3D to analyze the layouts which use an adaptive meshing mechanism in order to find a suitable mesh size automatically. The evaluation speed gain of the parasitic extraction models over Q3D allow the models to be used in an optimization setting.

Chapter 4 Multi-Objective Optimization

4.1 Introduction

The general goal of optimization is to minimize or maximize an attribute of a system by varying some set of design variables which describe the configuration or geometry of the system. The attribute of a system which is minimized or maximized is commonly referred to as an objective function or performance measure. For example, the performance measure for a fuel efficient vehicle may be the miles per gallon of gasoline a vehicle consumes, and the design variables may be the number of cylinders and cylinder volume of the vehicle's engine. In order to maximize fuel efficiency, the objective would be to maximize the miles per gallon (MPG) performance measure. The number of cylinders and cylinder size may be both decreased to maximize the MPG of the vehicle, but at the same time vehicle acceleration may suffer to such an extent, the vehicle is no longer a realistic form of transport. The vehicle acceleration also needs to be taken into account in order to create usable vehicles. In most engineering problems, it is necessary to consider more than one objective function or performance measure in order to gauge the utility of particular solutions. This form of optimization is referred to as multi-objective optimization as opposed to single objective optimization [22].

Multiple objective functions can be considered through single objective optimization through the use of composite objective functions where multiple performance measures are added together in a linear weighted polynomial:

$$F = a_1F_1 + a_2F_2 + \cdots a_nF_n, \tag{4.1}$$

where F is the composite objective function, F_i are the individual performance measures or objective functions, and a_i are the weights on each objective function. This method is called

linear scalarization [23], [24]. The weights on each performance measure need to be specified before the optimization is conducted in this case. In the vehicle example, the MPG and acceleration ratings would be two different performance measures F_1 and F_2. The individual weights a_1 and a_2 relative to each other determine whether an optimal solution favors higher MPG or higher acceleration. A trade-off exists between the two performance measures. A user of the linear weighting scheme usually will not know what real amount of trade-off exists between measures beforehand, thus prompting the user to select weights based on output from repeated optimization runs. This is considered an a priori type optimization scheme. A posteriori multi-objective optimization approaches skip this manual exploration process and instead seek to produce a trade-off curve directly [22], [24].

Multi-chip power modules also require multiple objective functions in order to gauge the utility of particular solutions and designs. The thermal and electrical parasitic models presented in the two previous chapters represent ways of measuring the performance of particular MCPM solutions. Previous research on MCPM design optimization also reflects the necessity of multiple objective functions for solution evaluation, thus multi-objective optimization is used for design exploration [25]. It is also advantageous to use a posteriori approaches of multi-objective optimization, because this allows a user to select from a set of trade-off solutions.

4.2 Pareto Frontiers and Multi-Objective Optimization

A posteriori multi-objective optimization generally produces a spectrum of solutions to a problem rather than a single solution as in single objective optimization. This spectrum of solutions represents a trade-off curve or Pareto frontier (Fig. 4.1). The primary goal in a posteriori multi-objective optimization is to find the actual trade-off curve which describes the

relation between objective functions which are in conflict, such as the maximization of a vehicle's MPG and acceleration [22]. In the case of MCPMs, a common objective conflict is between the maximum temperature in a module and the parasitic loop inductance. Typically, the maximum temperature and parasitic inductance both need to be minimized. Smaller sized modules tend to have lower parasitic inductance due to shorter path lengths of traces, but higher temperatures. Larger sized modules tend to higher parasitic inductance and lower temperatures, thus a conflict exists between the two objectives.

A contrived example of a trade-off curve between maximum temperature and parasitic inductance can be seen in Fig. 4.1. Individual solutions are represented by square boxes. Each individual solution represents a particular design and layout of a power module. An individual solution is represented by a vector of numbers where each number defines a geometric size of trace, positions of components in a module, etc. (the design variables). In most power module designs, there may be 10 to 20 or more different design variables present. The values in which the design variables take on do not appear in a Pareto frontier plot. A Pareto frontier plot only shows the relationship between a set of designs and the chosen set of objective functions.

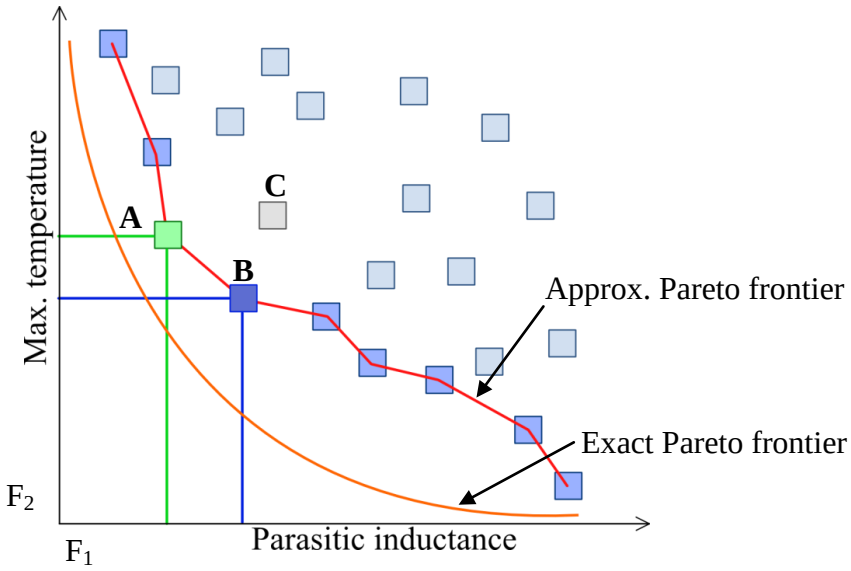


Fig. 4.1: Example of a Pareto frontier (trade-off curve).

All of the solutions which lie along the red curve in Fig. 4.1 are Pareto optimal solutions. A Pareto optimal solution is one in which no other solution in the set is able to dominate. For minimization type optimization problems, one solution dominates another solution when all of its objective functions are less than or equal to the other solution's and at least one objective function is strictly less than one of the other solution's objective functions. This can be stated mathematically as

$$F_i(x_1) \leq F_i(x_2) \;\; \forall \;\; i \in \{1,2,\dots,N\} \text{ and} \tag{4.2}$$

$$F_j(x_1) \leq F_j(x_2) \text{ for at least one } j \in \{1,2,\dots,N\} \tag{4.3}$$

where F_i and F_j represent each objective function in the optimization problem, x_1 and x_2 are the two solution design variable vectors being compared, and N is the total number of objective functions [22]. All of the solutions which do not meet criteria 4.2 and 4.3 are deemed dominated solutions and are usually not kept for post analysis as they are not Pareto optimal, but can be seen as light blue boxes in Fig. 4.1.

For instance (Fig. 4.1), solution C is dominated by at least solutions A and B, thus it is not Pareto optimal. Solution C not only has a higher temperature than solution A, but it also has higher inductance. According to this information, solution C has no advantage over solution A. Solution B on the other hand may have higher inductance, but it has lower temperature than solution A. Thus, there exists a valid trade-off between solution A and solution B.

The Pareto frontier illustrated in Fig 4.1 has some finite amount of solutions represented as square boxes. Some of these solutions are Pareto optimal relative to the given example set. These solutions are produced by a multi-objective algorithm which varies the design variables and evaluated the objective functions in search of optimal solutions. The objective functions in which the design variables are passed into are defined by the thermal and electrical models developed in the previous chapters. The design variables for a problem are mapped to a physical geometry of a module, and next the geometry is passed to the models in order to produce the objective function outputs (thermal and electrical measurements). Therefore, variations in the design variables are mapped to variations in the objective functions.

If the thermal and electrical parasitic models were simple enough to be represented by closed-form equations, the actual Pareto front could be possibly solved for analytically which would yield an exact analytical expression for the Pareto front [22]. Fig. 4.1 illustrates a comparison between a hypothetical exact Pareto front and an approximated front. Since the thermal and electrical models consist of complex algorithmic steps, solving for an analytical or exact solution set is not an option. The objective functions in this case are too algorithmically complex for analytical representation. Thus, evaluation based multi-objective algorithms are

needed to solve the optimization problems encountered in this thesis. Overall, an a posteriori and evaluation based multi-objective optimization algorithm is required for the software tool.

4.3 NSGA-II

The a posteriori multi-objective optimization algorithm selected for the MCPM layout optimization process is the nondominated sorting genetic algorithm II (NSGA-II) [26]. NSGA-II is a multi-objective evolutionary algorithm (MOEA) and is selected for its high regard in the optimization research community [27]. NSGA-II is based on a genetic algorithm, a subset of evolutionary algorithms. Genetic algorithms attempt to mimic the process of natural selection. Evolutionary algorithms in general attempt to mimic the processes of biological evolution. Since NSGA-II is based on a genetic algorithm, it is able to work with the thermal and electrical objective functions used in the MCPM layout synthesis tool. Genetic algorithms do not require the objective functions to be in any particular type of mathematical form. The objective functions can contain algorithmic steps and are not required to be closed-form equations.

The implementation of NSGA-II used in the MCPM layout synthesis tool comes from the distributed evolutionary algorithms in Python (DEAP) programming library [28]. DEAP contains implementations of many single objective algorithms, but currently NSGA-II as the only multi-objective algorithm. The DEAP implementation of NSGA-II requires 9 parameters: a list of design variable initial value and constraint intervals, the number of objective functions, a pointer to a function which returns all of the objective function values, the number of generations in which to run the algorithm, generational population size, offspring population size, an integer seed, the mutation probability, and the cross-over probability. These parameters are passed to

the constructor of a class object which is able to perform the optimization procedure and after optimization holds the solutions which represent the Pareto front.

Each design variable in the optimization problem is assigned an interval for initialization and one for constraint. An interval in this context is represented by a minimum and maximum value. At initialization, a population of the first generation of individuals is formed. Each individual is represented by a design variable vector. The numbers in the design variable vector are floating point values (approximately real). Each design variable is initialized with a random value using a uniform distribution over its given interval. After initialization, each design variable is confined to its given constraint interval. If a constraint interval is left to be null or "None" in the Python language, the design variable is left unconstrained.

The objective functions are passed to the optimization object via a function pointer or reference. A single function reference is sufficient for representing multiple objective functions. The function which is passed is required to return a vector with the same number of values as there are objective functions. The function is also required to take a design variable vector as input. In the case of the MCPM layout synthesis tool, any thermal or electrical parasitic model evaluations take place inside of this function.

The NSGA-II algorithm runs for some specified number of simulated generations of individuals. The genetic information or "DNA" of the individuals is represented by the design variable vectors. A constant population size is maintained for each generation. After the first generation is created through uniformly distributed random numbers, the best fit individuals in the population are selected for reproduction. The best fit individuals are iterated over and reproduced via mutation or cross-over until the off-spring population limit is reached. Mutation

and cross-over are genetic operators. Mutation randomly perturbs the values in the design variable vector and cross-over randomly exchanges values between two individuals' design variable vectors. The mutation and cross-over probability coefficients determine how often individuals are reproduced by the respective genetic operators. The best fit individuals from the offspring population are selected and go on to replace lesser fit individuals in the previous generation. After replacement, this population becomes the next generation of individuals and the cycle is repeated.

The NSGA-II algorithm rates best fit individuals through a process which measures crowding on the Pareto frontier. Individuals which reside in less crowded sections of the Pareto front will have higher fitness than individuals in the crowded sections. NSGA-II also uses a fast sorting algorithm for determining whether new individuals entering the population are nondominated (Pareto optimal). The faster sorting technique is one of the main improvements over the first NSGA algorithm. The crowding fitness and fast sort mechanisms are the primary characteristics of NSGA-II [26].

The next chapter covers the process in which the design variables for a layout are extracted from a symbolic layout. These are the design variables which are varied by the optimizer in search of Pareto optimal solutions.

Chapter 5 Symbolic Layout

5.1 Introduction

A symbolic layout represents the form of an MCPM layout without reference to specific geometric sizes, scales, component positions, or component types. This allows a symbolic layout to be used across multiple projects and allows for optimization through geometric layout variation. A symbolic layout is similar to a "stick diagram" used in the initial design phases of digital logic gate layouts in integrated circuits (ICs), but the meaning of the elements used in MCPM symbolic layouts differs substantially from those in the IC domain. The symbolic layout concept was first developed by Neil Weste at Bell Labs for representing IC layouts in a process agnostic form [29]. A set of design rules for a process are applied to the symbolic layout yielding a geometric layout. A similar set of processes are adapted for MCPMs, but instead of applying a fixed set of design rules to yield a layout, a parameterized layout is generated. These are the parameters or design variables in which the optimizer varies to obtain a set of solutions.

Fig. 5.1(a) demonstrates an MCPM symbolic layout, and Fig. 5.1(b) and Fig. 5.1(c) are two different geometric layouts derived from it. Symbolic layouts are composed of a set of lines and points. For MCPMs, the lines represent either traces or bondwires, and the points represent either package leads or semiconductor devices.

Lines are only allowed to run in horizontal (x-axis) or vertical (y-axis) directions. Any line which is not identified as a bondwire by default represents a rectangular segment of trace with a width value associated with it. A design variable is associated with each of these width values. The width is always orthogonal to a trace's orientation, for example a vertical trace's

width is in the horizontal direction, and a horizontal trace's width is in the vertical. A scanning type process is described in more detail in a later section as to how the design variables are allocated for trace sizing.

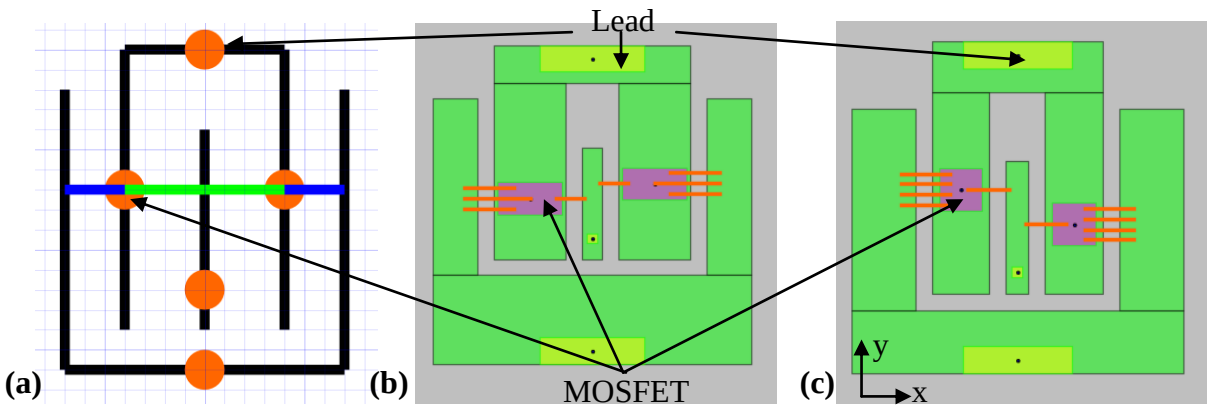


Fig. 5.1: Symbolic layout with two different geometric layouts. (a): Symbolic layout. (b): Geometric layout 1. (c): Geometric layout 2.

Each point element in a symbolic layout is associated with a specific type of semiconductor device or lead selected by a user. Each type of semiconductor device or lead has dimensions and material properties associated with it through a technology data structure. A user is able to create new device and lead technology entries through the Technology Library Editor interface which is detailed in Chapter 7. The semiconductor devices are free to move one dimensionally along the piece of trace they reside on. For example, if the piece of trace is vertical, the die is able to move up and down vertically within the bounds of the trace. A design variable is allocated for each die's placement on its respective trace. On the other hand, the point elements identified as leads are not given a design variable. These are given a fixed relative position with respect to the trace such as the middle, top, or bottom.

Bondwires also need representation in order to connect devices to trace or even trace to trace. The two supported types of devices are diodes and MOSFETs. Diodes are required to have

one trace connection via bondwire (the anode) and MOSFETs two (the gate and source). Symbolic lines are used to represent these connections. No design variables are associated with these connections, and the geometric placement of bondwires is dependent on die and trace placement.

The next sections of this chapter will detail the loading of symbolic layouts, the extraction of design variables from symbolic layouts, and the generation of layout geometry given a set of design variable values. The layout generation process consists of two major steps: trace layout generation and component placement.

5.2 Symbolic Layouts and Matrix Representation

A symbolic layout is stored as a scalable vector graphic (SVG) file. SVG files use extensible markup language (XML) to describe vector based graphics or drawings. A vector graphic is described by a set of floating point numbers as opposed to pixels used in raster or bitmap graphics. A line for example is described by two points P_0 and P_1 where $P_i = \{x, y\}$ (a 2D coordinate). A line is represented by "path" XML element which contains only two 2D coordinates. A "path" element may have more than two coordinates for SVG drawing capabilities, but these "path" elements are ignored if present. The points in a symbolic layout are represented by "arc" XML element tags which describe arcs or circles in SVGs. Each "arc" element contains a 2D coordinate for a center point of an arc or circle. This center position is extracted to represent the position of a point in a symbolic layout.

Every "arc" and "path" element in the SVG file is parsed and stored in a "LayoutPoint" and "LayoutLine" data structure, respectively. The "LayoutPoint" and "LayoutLine" data

structures are Python class objects with data fields to store the element coordinates. A list containing both types of data structures is stored. This list constitutes a loaded symbolic layout. Only purely vertical and horizontal lines are allowed in the layout. Next, the line and point coordinates which make up the symbolic layout are normalized.

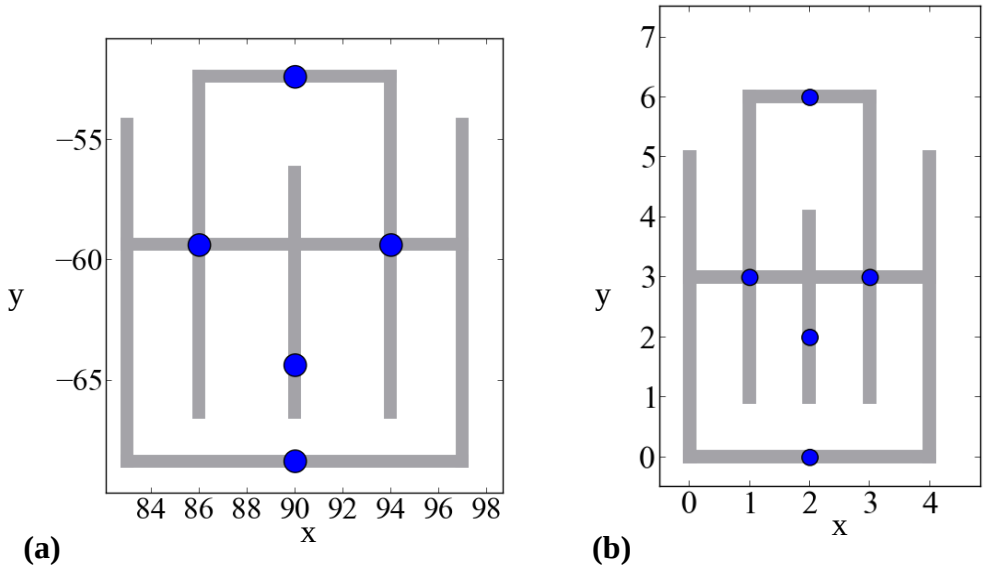


Fig. 5.2: Symbolic layout normalization. (a): Un-normalized layout. (b): Normalized layout.

The normalization process removes any arbitrary scales and translation from a symbolic layout and allows for construction of a matrix data structure of the layout. For example, the un-normalized layout in Fig. 5.2(a) spans around 14 units on the x and y axes and its origin arbitrarily located. The normalized layout in Fig. 5.2(b) spans exactly 4 by 6 units on the x and y axes with an origin at (x=0, y=0). All coordinates in a normalized layout take on non-negative integer numbers.

The normalization process is started by creating a list of all x coordinates L_x and a separate list of all y coordinates L_y held in the point and line layout objects. Any duplicate x or y coordinates are removed from the lists. The two lists represent all unique x positions and y

positions in a layout. Take for example $L_x = \{8.9, 3.1, 5.6\}$ and $L_y = \{2.2, 5.1, 4.8\}$. The lists are sorted from least to greatest e.g. $Sort(L_x) = \{3.1, 5.6, 8.9\}$ and $Sort(L_y) = \{2.2, 4.8, 5.1\}$. After sorting, two new lists I_x and I_y are created which are enumerated with integers from 0 to N-1 and 0 to M-1. N and M are the list lengths of L_x and L_y, respectively. $I_x = \{0, 1, 2\}$ and $I_y = \{0,1,2\}$ given the example L_x and L_y. A one-to-one mapping exists between the lists (L_x and I_x) and (L_y and I_y). Every unique x and y coordinate now has a non-negative integer representation. A new list of point and line objects is created with the normalized coordinate system by replacing the un-normalized layout coordinates with the normalized coordinates via the one-to-one mapping.

After the layout is normalized, it is transformed into a matrix representation. Each coordinate in the normalized layout corresponds to an entry in a 2D matrix. Fig. 5.3(a) and Fig. 5.3(b) show a small normalized layout and its corresponding matrix representation, respectively. The entries of the matrix each contain a list of references to the layout objects where are present at the corresponding coordinate in the symbolic layout. For example, the entries which contain the label "Trace1" in Fig. 5.3(b) mean they contain a reference to a "LayoutLine" object. The names given to the objects in Fig. 5.3 are for illustrative purposes only. These objects are referred to by pointers (memory locations) in the actual program.

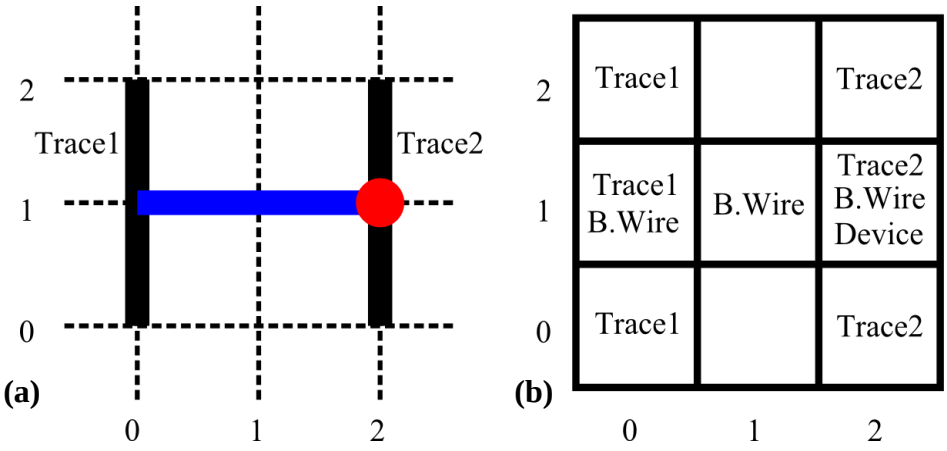


Fig. 5.3: Normalized layout to matrix representation. (a): Normalized layout. (b): Matrix data structure with pointers to layout objects.

Another symbolic layout containing only "LayoutLine" objects identified as traces is constructed. The "LayoutLine" objects which are identified as bondwires are excluded. The normalization process is conducted on the trace specific layout and a corresponding matrix is created. The symbolic layout in Fig. 5.3(a) reduces to a 2x2 trace specific matrix. The trace specific layout and corresponding matrix is shown in Fig. 5.4. The trace specific matrix is used extensively in the design variable extraction process detailed in the next section.

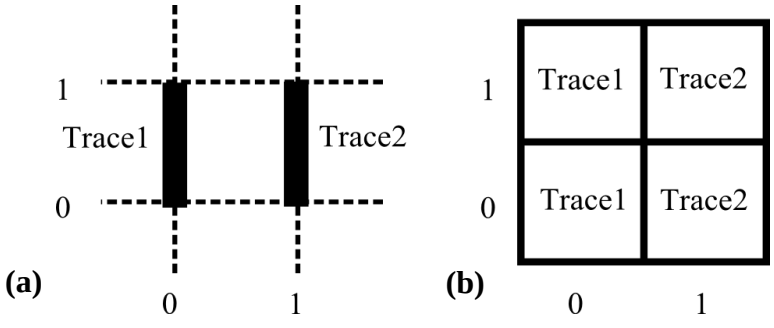


Fig. 5.4: Normalized symbolic layout and matrix with only traces. (a): Normalized trace specific layout. (b): Trace specific matrix data structure.

5.3 Layout Design Variable Extraction

The optimization engine needs to know how many design variables there are in a design problem, thus total number of design variables used in a layout needs to be identified. As mentioned earlier, a single design variable is assigned for positioning each semiconductor device. This count is trivial. The rest of the design variables stem from the trace specific layout, the portion of the layout concerning only traces. The design variables are counted by analyzing the trace specific matrix. In order to explain this process, a simple trace layout is analyzed first (Fig. 5.4) which helps form the basis for the general analysis algorithm.

The simple trace layout shown in Fig. 5.4 is composed of two vertical trace segments. Fig. 5.5 shows two corresponding geometric layouts derived from the symbolic layout in Fig. 5.4. The substrate width and length, the ledge width, and the gap width are all held constant over the optimization process, so no design variables are assigned for these dimensions (Fig. 5.5). All trace is inset around the border of the substrate by the ledge width and each trace is ensured to be separated by at least the gap width. The traces in a layout are assumed to expand as much as possible to fill the substrate as much as possible. Trace1 and Trace2 are both vertical traces which span the entirety of the symbolic layout in the vertical direction. There are no other trace objects above or below the Trace1 or Trace2. This means that Trace1 and Trace2 will automatically span the geometric layout vertically, and no design variables are assigned for the vertical portion of the problem.

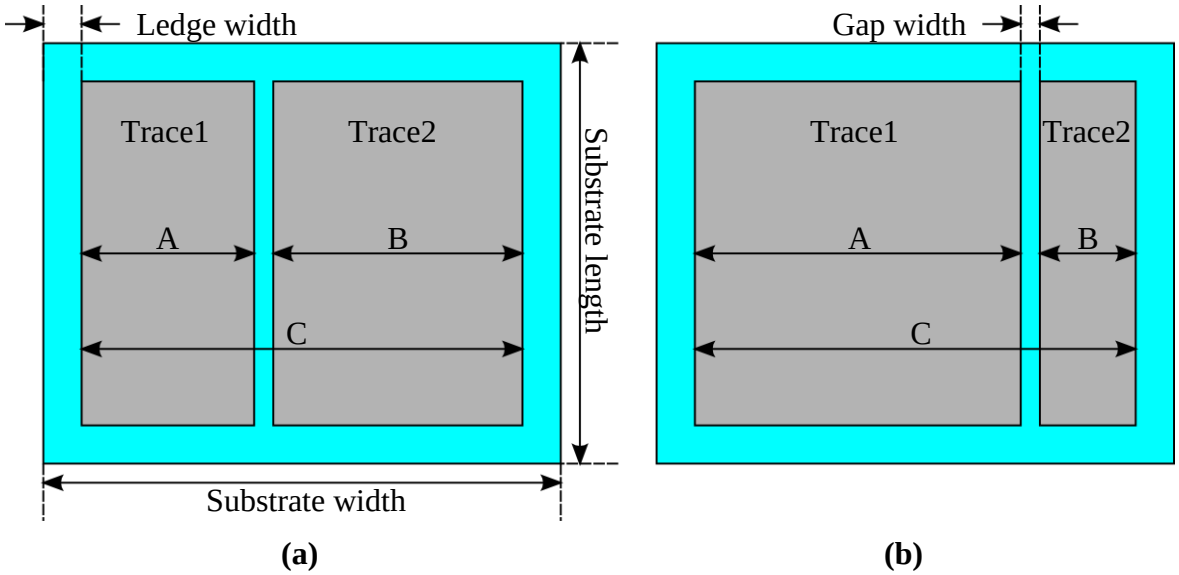


Fig. 5.5: Two trace layouts derived from the same symbolic layout. (a): Trace layout 1. (b): Trace layout 2.

Looking at the problem horizontally, there are two variable width traces. Trace1 and Trace2 cannot span the layout in the horizontal direction simultaneously. The design variables denoted A and B may be assigned to the widths of Trace1 and Trace2. One design variable can be removed; because the overall width of the layout is fixed. If the design variable A is removed, A is calculated given B e.g. $A = f(B) = (C - G) - B$. The removal of a design variable not only reduces the design space by a dimension, but also allows for a better chance of producing viable solutions. The optimizer would rarely be able to randomly find values of A and B which exactly sum to $(C - G)$. This removal process is also applied in the vertical direction. In this case, no design variables are needed in the vertical direction.

The design variable analysis algorithm is broken into a horizontal scan and a vertical scan. The horizontal scan looks at each column in the trace specific matrix and the vertical scan at each row. The same algorithm is used for both horizontal and vertical scans. The pseudocode for the horizontal scan is as follows:

```
Input Parameter: trace_matrix                    # Trace specific matrix (2D) input
dv_count = 0                                     # Init. design variable count to zero
column_list = { }                                # Create empty column list
N = x_length(trace_matrix)                       # Number of columns in trace matrix
M = y_length(trace_matrix)                       # Number of rows in trace matrix

for x = 0 to N-1:                                # Iterate over all columns in matrix
    # Build list of unique vertical traces in current column
    vert_traces = { }                            # Holds unique vertical trace objects
    for y = 0 to M-1:                            # The length of the column is M
        for each trace in trace_matrix[x][y]:    # Get all traces in current position
            if  (trace is vertical) and (trace not in vert_traces):
                # Trace is unique and vertical, append to vert_traces list
                vert_traces.append(trace)

    dv_indices = { }                             # Indices of design variable list
    if length(vert_traces) == 0:                 # Empty col. (Phantom)
        # Column represents empty space,
        # Assign a design variable to width of column
        dv_indices.append(dv_count)
        dv_count = dv_count + 1
    else if length(vert_traces) >= 1:            # Single or multiple trace column
        # Give each vertical trace object in the column a design variable
        for each trace in vert_traces:
            dv_indices.append(dv_count)
            dv_count = dv_count + 1
    column = {vert_traces, dv_indices}           # Create column object
    column_list.append(column)                   # Append column object to list
```

The two primary pieces of data which are determined by the scan are the **dv_count** and **column_list**. The **dv_count** is a count of all of the design variables found in the scan process. The **column_list** is a list of **column** objects where each **column** object consists of two data fields, **vert_traces** and **dv_indices**. The **vert_traces** field is a list which stores all of the unique vertical traces in the column. Only vertical traces are examined in the horizontal scan. The **dv_indices** field is a list which stores all of the assigned design variable indices in the column. The design variable indices are integer numbers which are indices to the design variable vector.

The design variable vector is created after executing both the horizontal and vertical scans after the total design variable count is known.

For example, Fig. 5.6 has a total of 4 columns in the trace specific matrix. Column 0 has only one unique vertical trace: T1. T1 appears twice in this column, but its second instance is excluded from the **vert_traces** field for the column. T2 also appears in the row 1 of the column, but T2 is not vertical, so it is not included either. Since the column has one unique vertical trace, a single design variable is assigned to the column. If more than one vertical trace existed in this column, a design variable is assigned for each trace.

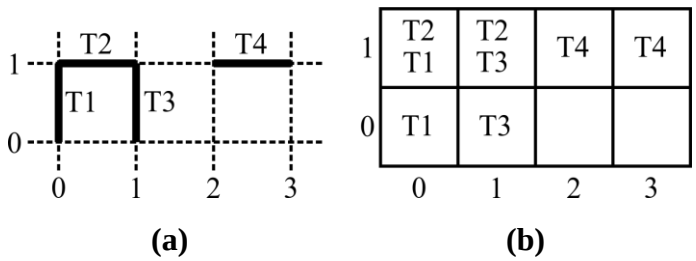


Fig. 5.6: A slightly more complicated trace specific layout. (a): Trace specific layout. (b): Trace specific matrix.

Column 1 in Fig. 5.6(b) contains one unique vertical trace also, which is T3. Columns 2 and 3 contain no unique vertical traces. Only the horizontal trace T4 runs across row 1 of these two columns. A column which contains no vertical traces is named a phantom column. The phantom columns still have geometric significance in the form of spacing. Each column is assigned a design variable to represent this spacing. In this particular example, the length of the horizontal trace T4 is determined by these horizontal spacing design variables or phantom columns. The total design variable count **dv_count** for the horizontal scan in this example

amounts to 4. The T1 and T3 widths and 2 phantom column widths are all determined by these 4 design variables.

After the horizontal scan, a design variable is removed. This portion of the algorithm is not shown in the previous pseudocode. Any of the design variables allocated during the horizontal scan of the current example are considered removable variables, because each fully determines the variable width of their respective column. Not all design variables are removable. A design variable is considered removable if it is the only variable allocated for a column or row. If multiple unique trace elements exist in a column or row, none of the design variables allocated in the column or row are considered removable. A removable variable is found by iterating over the **column_list** (starting at the column 0) and finding the first **column** with only one design variable allocated e.g. length(**column**.**dv_indices**) == 1. In this example, column 0 is found to contain one design variable, the width of T1, whose index is flagged for removal. In reality, this index will remain part of the design variable vector, but the value at the index is ignored. The optimizer may set values for the number at this index, but it will not affect design geometry. If a removable design variable is not found, the layout is considered "stiff" and an error is thrown.

The vertical scan of rows in the trace specific matrix is extremely similar to the horizontal scan of columns. A one-to-one change of variables and operators of the horizontal scan algorithm produces the vertical scan algorithm. The vertical scan pseudocode can be found in Appendix E. The **dv_count** variable is initialized to the value of **dv_count** where the horizontal scan left off. The **column_list** is replaced with **row_list**, which holds a set of **row** objects instead of **column** objects. **N** and **M** are swapped to represent the number of rows and columns, respectively. The **vert_traces** list is replaced with **horz_traces** which holds all unique

horizontal trace elements in a row. The loop which identifies unique vertical traces is changed to identify unique horizontal traces. The rest of the algorithm is left unchanged.

The example shown in Fig. 5.6 contains two rows. Row 0 contains no horizontal trace elements, thus it is allocated a single design variable which represents a spacing width. The widths in the vertical scan case run vertically (along the y-axis) or orthogonal to the widths in the horizontal scan. Row 1 contains two horizontal traces T2 and T4, thus two design variables are assigned for this row. The design variables in this row are not removable, because neither variable always determines the width of the entire row. For example, T2 may be wider than T4 in some instances or vice versa. A total of 3 design variables are allocated during the vertical scan.

The design variable removal process for the vertical scan is the same as the horizontal scan. **row_list** is iterated over until a row containing only one design variable is found. This design variable is indexed and ignored during the layout generation process.

A geometric instance of the trace layout example is shown in Fig. 5.7. The 4 design variables found in the horizontal layout scan are labeled A, B, C, and D. The 3 design variables found in the vertical scan are labeled E, F, and G. Considering only the trace layout, the design variable vector is 7 entries long e.g. **dv_vector** = {A, B, C, D, E, F, G}. If any devices were present in the layout an extra variable would be associated with each. The variables at positions 0 and 4 (A and E) are removed or ignored during the geometric layout generation algorithm. Note that in Fig. 5.7, row 1 has two design variables F and G. F is larger than G in this particular geometric layout, thus currently defines the width of row 1. It is possible that the inverse of this could possibly happen, thus neither F nor G always define the width of the row. There is insignificant information to algebraically solve for the width of the row based on other variables.

For this reason, F and G are considered non-removable design variables. The process which generates geometric trace layouts such as Fig. 5.7 is elaborated in the next section.

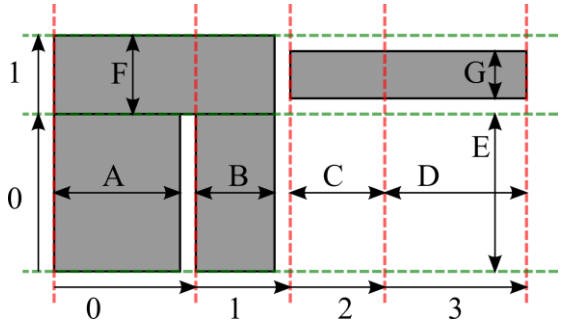


Fig. 5.7: Geometric trace layout example with design variables.

5.4 Trace Layout Generation

The trace layout generation process produces a set of rectangular trace segments given a set of design values (the design variable vector) and the row and column lists created in the previous section. The trace layout generation process is executed many times throughout the optimization process while the design variable analysis is only executed once. The generation process is divided into a horizontal and vertical set of processes similar to the design variable analysis. The set of processes is identical for the horizontal and vertical axes except for a change of variables and operators. The horizontal and vertical generation processes are each broken into three steps or passes:

1. Conservation pass: Solve for the removed design variable width.
2. Parallel generation pass: Generate rectangle coordinates for trace elements whose widths are parallel to the pass direction.
3. Orthogonal generation pass: Generate rectangle coordinates for trace elements whose widths are orthogonal to the pass direction.

The first pass involves finding the width of the removed design variable in either the horizontal or vertical case. The value in the design variable vector which corresponds to the removed design variable is ignored, so the correct value is solved for based on the un-removed design variables and fixed substrate dimensions. It is called the conservation pass, because the total dimensions of the substrate must be conserved over variation of the design variables.

The second and third steps involve setting the rectangular coordinates (top, bottom, right, and left) for the trace segments. For the horizontal case, the second pass involves setting the left and right (x-axis) coordinates of the rectangles which represent the vertical trace elements. The third pass involves setting the left and right coordinates of the rectangles which represent horizontal trace elements. For the vertical case, the second and third passes involve setting the top and bottom (y-axis) coordinates of the rectangles which represent the horizontal and vertical trace elements.

The pseudocode for the horizontal case of the first pass is as follows:

```
Input Parameters: column_list                    # List of column objects
                  removed_dv_index               # Removed design variable index
                  gap_width                      # Width between traces
                  dv_vector                      # Vector (list) of design values
                  layout_width                   # Total layout width (fixed)

known_width = 0.0                                # Accumulates total known width
N = length(column_list)                          # Number of columns
for i = 0 to N-1:                                # Iterate over all columns
    column = column_list[i]                      # Get column object from list
    phantom = (length(column.vert_traces) == 0)  # True if column is phantom

    # Determine if column has a trace gap
    if i < N-1 and not phantom:                      # Not right bound column, not phantom
        known_width = known_width + gap_width        # Add gap widths

    # Find maximum design variable width in column
    max_width = 0.0
    for each dv_index in column.dv_indices:      # Iterate through all dv indices
        if dv_index != removed_dv_index:         # Cannot be a removed variable
            width = dv_vector[dv_index]          # Retrieve design value
            if width > max_width:                # Check if larger than max_width
                max_width = width                # Set new max_width

    known_width = known_width + max_width        # Add max. column width
removed_width = layout_width – known_width       # Solve for removed dv width
```

The primary goal of the conservation pass is to find the width of the removed design variable for either the horizontal or vertical axis. In order to generate the coordinates of the trace rectangles, the two removed design variables need to be solved for. In the previous example, the design variables A and E are removed (Fig. 5.7). The layouts shown in Figs. 5.8(a) and 5.8(b) are a continuation of this example.

In the horizontal case, the design variable A is found by subtracting the known width U from the total layout width O i.e. $(A = O - U)$. In the pseudocode, U is represented by

known_width, O is **layout_width**, and A is **removed_width**. **known_width** is a sum of all the known widths in a layout (on either axis). Each column in the layout has some width which is a combination of a gap width (or not) and the maximum design variable value (**max_width**) in the column. Every column in a layout has a gap spacing of **gap_width** on its right side except for the last column (N-1) and any phantom columns. Column 0 and 1 both have gaps, but columns 2 and 3 do not (both are phantom). The **max_width** of column 0 is not known beforehand, thus it is not included in the **known_width** value, but the gap of column 0 is known and it is included. The widths of columns 1, 2, and 3 are fully known and are included in **known_width**. After **known_width** is summed, **removed_width** is calculated. This concludes the horizontal case.

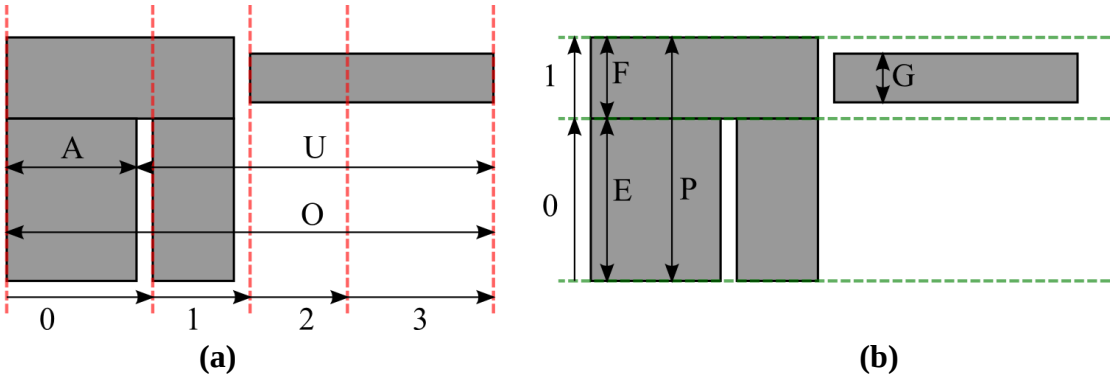


Fig. 5.8: Finding removed design variable widths. (a): Horizontal case where A is found. (b): Vertical case where E is found.

The same process is completed in the vertical case except operating on rows instead of columns. The design variable E is an unknown value and needs to be solved for. The contribution to **known_width** for row 0 is nothing, because it is phantom (no gap) and its **max_width** value is determined by the removed design variable E. Row 1 contributes a **max_width** of F to **known_width** and has no gap (last row). Row 1 contains two design variables F and G. In this example, the width of the row 1 is determined by F, because G is

smaller than F. The **removed_width** or E is equal to $P - F$ where P is the overall vertical width of the layout. This concludes the vertical conservation pass.

After values for the removed design variables are found, the next two passes generate the actual coordinates for the trace rectangles. Each trace rectangle is defined by four coordinates: left, right, bottom, and top. The parallel generation pass in the horizontal case generates the left and right (x-axis) coordinates for the vertical trace elements and in the vertical case it generates the bottom and top (y-axis) coordinates for the horizontal trace elements. The orthogonal generation pass in the horizontal case generates the x-axis coordinates for horizontal elements and in the vertical case generates the y-axis coordinates for the vertical elements. The orthogonal passes (horizontal and vertical) yield the lengths of the trace rectangles.

The pseudocode for the parallel generation pass for the horizontal case is as follows:

```
Input Parameters: column_list                                  # List of column objects
                  removed_dv_index                             # Removed design variable index
                  gap_width                                    # Width between traces
                  dv_vector                                    # Vector of design values
                  removed_width                                # Removed variable width

prev_column_right = 0.0                                        # Start column coordinates at 0.0
N = length(column_list)                                        # Number of columns
for i = 0 to N-1:                                              # Iterate over all columns
    column = column_list[i]                                    # Get column object from list
    column.left = prev_column_right                            # Set left coordinate of column
    removed = (column.dv_indices[0] == removed_dv_index)       # True if dv for column is
                                                               # removed
    num_elements = length(column.vert_traces)                  # Number of vertical traces in col.

    if removed:
        dv_index = column.dv_indices[0]                        # Index of first variable
        dv_vector[dv_index] = removed_width                    # Replace width value in vector

    if num_elements == 0:                                      # Phantom column (no traces)
        dv_index = column.dv_indices[0]                        # Get column dv vector index
        width = dv_vector[dv_index]                            # Retrieve width from dv vector
        column.right = column.left + width                     # Set right coordinate of column
    else if num_elements > 0:
        max_width = find_max_width(column_list, dv_vector)     # Find max dv width in col.
        column.right = column.left + max_width                 # Set right hand coordinate
        column.mid = 0.5*(column.left + column.right)          # Find mid-point of column
        if i < N-1:                                            # If not the last column
            column.right = column.right + gap_width            # add gap width
        M = length(column.dv_indices)                          # Number of design vars. in col.
        for j = 0 to M-1:                                      # Iterate through traces in col.
            dv_index = column.dv_indices[j]                    # Get design var. index
            half_width = 0.5*dv_vector[dv_index]               # Get trace width from dv vector
            trace = column.vert_traces[j]                      # Get trace object
            trace.rectangle.left = column.mid-half_width       # Set left side of trace rect.
            trace.rectangle.right = column.mid+half_width      # Set right right side of rect.
    prev_column_right = column.right                           # Set previous right coordinate
                                                               # before next iteration
```

The parallel generation step for the horizontal case moves from left to right beginning at column 0 and ending at N-1. The variable **prev_column_right** stores the previous column's

right coordinate which gives reference for placing the coordinates of the current column. The left, mid, and right coordinates (Fig. 5.9(a)) are located for a column first. The left coordinate for a column is always equal to the right coordinate of its previous column. The right coordinate is determined by translating the left coordinate by the column's **max_width** and a **gap_width**, if the gap criteria are met. The mid coordinate is the left coordinate translated by half the column's **max_width**. This centers the mid coordinate on the trace with the largest width in the column. All of the smaller width vertical traces are centered on mid as well. After the column's coordinates are placed, each vertical trace in the column is given a left and right coordinate using its width value from the design variable vector. The removed design variable in the vector is also replaced with the **removed_width** value calculated in the previous pass, such that the correct width is given to the trace.

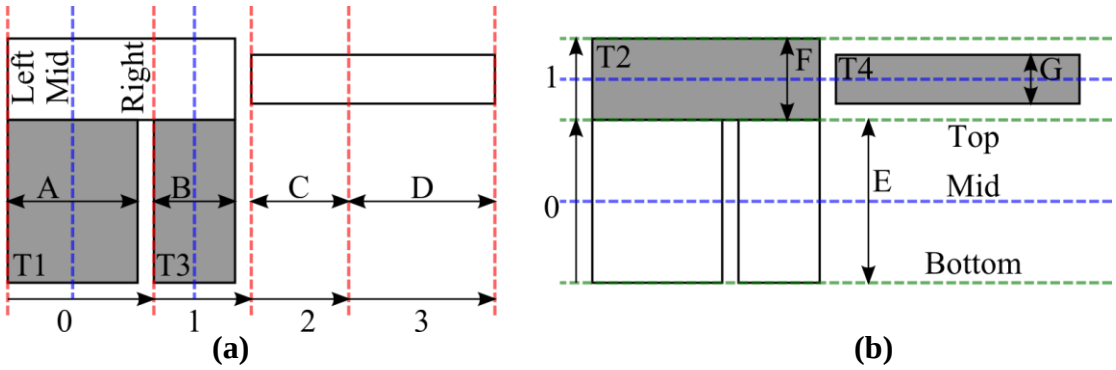


Fig. 5.9: Parallel trace rectangle generation. (a): Horizontal process. (b): Vertical process.

The vertical portion of the parallel generation pass iterates over the rows instead of columns and sets the top and bottom coordinates for the horizontal trace elements. These would be traces T2 and T4 shown in Fig. 5.9(b). The first row is a phantom row, because it lacks any horizontal trace elements, but it still has bottom, mid, and top row coordinates. These coordinates are determined by the design variable E found during the conservation pass. The second row

contains two horizontal trace elements. The bottom coordinate for row 1 is equivalent to row 0's top coordinate. The top coordinate is determined by translating the bottom coordinate by the **max_width** of the row. **max_width** for row 1 is equivalent to the design variable F (width of T2). There is no gap spacing for row 1 because it is the last row. The top and bottom rectangle coordinates for T2 and T4 are determined by their respective design variables and centered on T2 which is the widest trace in this instance.

After the parallel generation pass, every trace rectangle has a defined width. The orthogonal generation pass defines the lengths of trace rectangles, thus fully defining the dimensions of each rectangle.

The pseudocode for the horizontal case of the orthogonal generation pass is as follows:

```
Input Parameters: horz_trace_list          # List of all horizontal trace elements
                  column_list              # List of column objects

N = length(horz_trace_list)                # Number of horizontal trace elements
for i = 0 to N-1:                          # Iterate over all horz. traces
    trace = horz_trace_list[i]             # Get trace object
    left_index = trace.pt1.x               # Normalized x coord. of left point
    right_index = trace.pt2.x              # Normalized x coord. of right point

    left_coord = column_list[left_index].left   # Get left side of column at index
    trace.rectangle.left = left_coord           # Set left side of trace rect.

    right_coord = column_list[right_index].right # Get right side of column at index
    if right_index < N-1:                       # If not at the far right layout bound
        right_coord = right_coord - gap_width   # leave gap width of space
    trace.rectangle.right = right_coord         # Set right side of trace rect.
```

The horizontal orthogonal generation iterates through a list of all of the horizontal trace elements in the layout represented by the variable **horz_trace_list**. Each trace element is defined by two points in the normalized trace layout. The x coordinate of the first point (pt1.x) of a horizontal trace element is always less than the x coordinate of the second point (pt2.x) and both points share the same y coordinates. Each x coordinate in the normalized trace layout is associated with a column and each column has geometric coordinates (left, mid, right). This allows definition of the left and right geometric coordinates for the trace rectangle. The trace element T2 in Fig. 5.10(a) starts at column 0 and ends at column 1. The left trace rectangle coordinate is always placed at the left coordinate of the starting column. The right trace rectangle coordinate is placed at the right coordinate of the ending column. If a gap exists in the ending column, the right coordinate of the trace rectangle is reduced by **gap_width**. The horizontal trace rectangles are fully defined after this process.

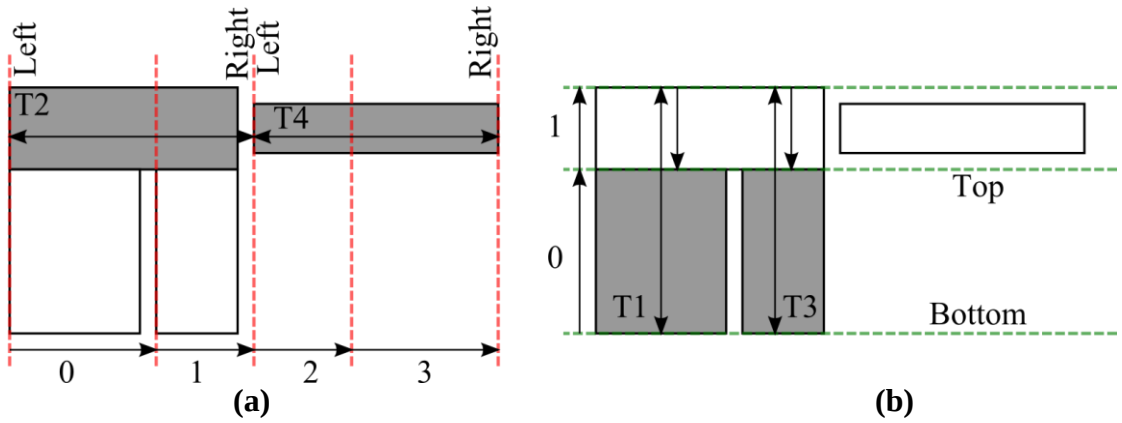


Fig. 5.10: Orthogonal trace rectangle generation process. (a): Horizontal process. (b): Vertical process.

The vertical case of the orthogonal generation pass operates on rows instead of columns and defines the top and bottom coordinates of the rectangles associated with vertical trace elements. The primary difference between the two algorithms is the interaction between horizontal and vertical trace elements. For example, the trace rectangles for T1 and T3 would both intersect with T2 if their top coordinates where placed at the top coordinate of row 1. If an intersection between trace elements is detected, the top coordinate of the vertical trace rectangles are changed to the bottom coordinate for the row (row 1 in this case). All trace rectangles in the layout are fully defined after this step.

5.5 Super Traces

The trace segments described in the two previous sections are all either horizontal or vertical lines which span between two x coordinates or two y coordinates in normalized trace specific layout coordinates. The lines cannot span multiple columns and rows simultaneously. A super trace is able to span multiple columns and rows, and it is represented by crossing a horizontal and vertical line in the symbolic layout (Fig. 5.11(a)). The same trace layout

generation process is executed and a layout such as Fig. 5.11(b) is generated. A rectangle is generated for the crossed vertical and horizontal lines as normal. After layout generation, any crossed lines are detected and transformed into a super trace. This is achieved by creating a new rectangle which takes of the top and bottom coordinates of the vertical trace and the left and right coordinates of the horizontal trace. Fig. 5.11(c) shows the outcome of this process. The next step in layout generation is component placement which is described in the following section.

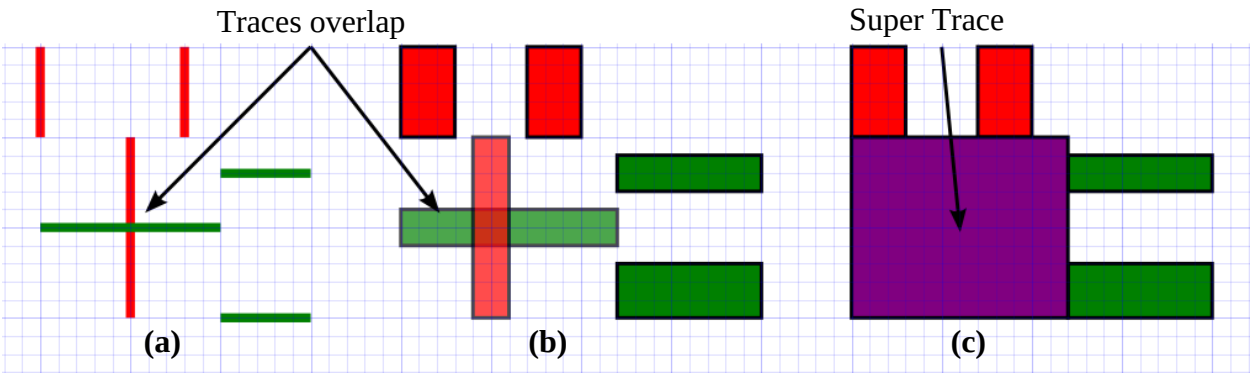


Fig. 5.11: Super trace formation. (a): Symbolic layout. (b): Overlapping rectangles after trace rectangle generation. (c): Super trace replacement.

5.6 Component Placement

The two types of components supported in the tool are devices and leads. The devices and leads are placed relative to the traces. Each device or lead is represented by a point in the symbolic layout and must reside on a trace line which is called its parent. The devices and leads cannot reside on a line which comprises a super trace for this version of the tool. Each device is assigned a design variable which is constrained to the interval $[0, 1]$. The value of this design variable defines where the center point of a device will be placed along the length of a trace. The center point of the device is constrained to the middle of the width of a trace. Fig. 5.12(a) shows

a symbolic layout with three vertical traces and a single device on the middle trace. Fig. 5.12(b) shows a corresponding geometric layout of Fig. 5.12(a) neglecting bondwires. The device is placed at 80% the length of its parent trace.

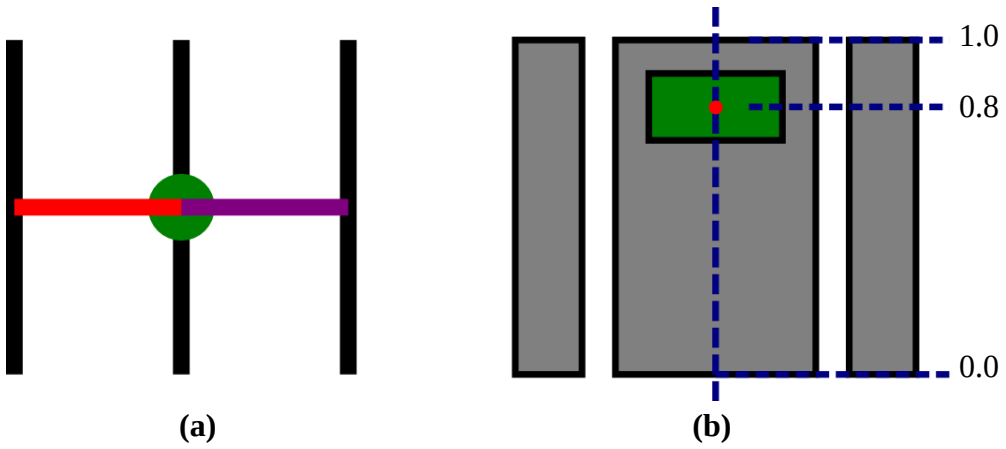


Fig. 5.12: Device placement. (a): Symbolic layout. (b): Geometric layout with device placed at 80% length of trace.

The bondwires which connect devices to traces also require placement. Each device has a set of power and signal bondwire connection points associated with it. These points are defined relative to the device and do not change over the course of optimization. MOSFET type devices require two symbolic bondwire connections: a source and gate connection. Diode type devices require only a cathode connection.

The symbolic bondwires are identified as power or signal type bondwires (by a user or possibly an algorithm). In Fig. 5.12(a), the bondwire line on the right is identified as a power type bondwire and on the left a signal type bondwire. A MOSFET device requires a power and signal bondwire connection for the source and gate, respectively. The bondwire landing points on a device are oriented based on the symbolic bondwire connections to the device. Fig. 5.13 is a close-up of Fig. 5.12(b) with bondwires included. The device is oriented such that the power

(source) bondwires are on the right side and the signal (gate) bondwire is on the left. If the symbolic power and signal bondwires were swapped, the device bondwire points would be rotated 180° around the device center point.

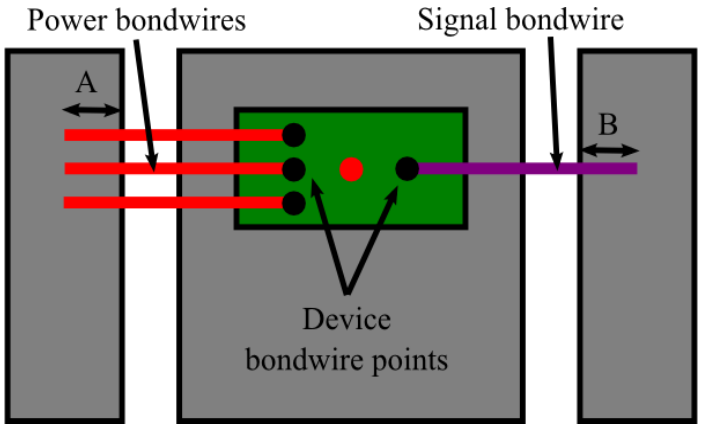


Fig. 5.13: Bondwire placement.

After orienting and placing the device bondwire points, the trace landing points are placed. Symbolic bondwires each start at a device and end on a trace element. Each trace element has an associated trace rectangle. The landing points are placed relative to the trace rectangle. Each device point has a corresponding trace landing point. For horizontal bondwires, the y coordinate for each device point is used for the trace landing points (same y location). The x coordinate is inset on the connected trace by a specific amount defined by a design rule, labeled as A and B in Fig. 5.13. The design rules for layouts are discussed in greater detail in the last section of this chapter. The vertical bondwires are placed in a similar manner, except the coordinate axes are exchanged. All of the generated bondwire points are saved in the symbolic bondwire element objects, such that the data is easily retrievable.

The final step for layout generation is the placement of package leads. A design variable is not allocated for each lead, but instead they are placed at fixed relative positions along the

length of traces. Leads are placed at either the right or left boundaries or centered along the length of horizontal traces, depending on where the symbolic lead point is placed on its parent line. Fig. 5.14(a) shows lead point L1 positioned in the middle of its parent line. Since L1 is detected to be near the middle of its parent line, it is kept centered (length wise) on its parent trace. L2 and L3 are positioned near the left and right points of their parent line, thus L2 and L3 are placed as close to the left and right trace rectangle boundaries as possible. The position of a lead along the width of the trace is determined by its symbolic position in the overall symbolic layout. If a lead is placed in the upper-half of a layout and resides on a horizontal trace, it is placed at the top of its parent trace rectangle. If placed in the lower-half, it is placed at the bottom.

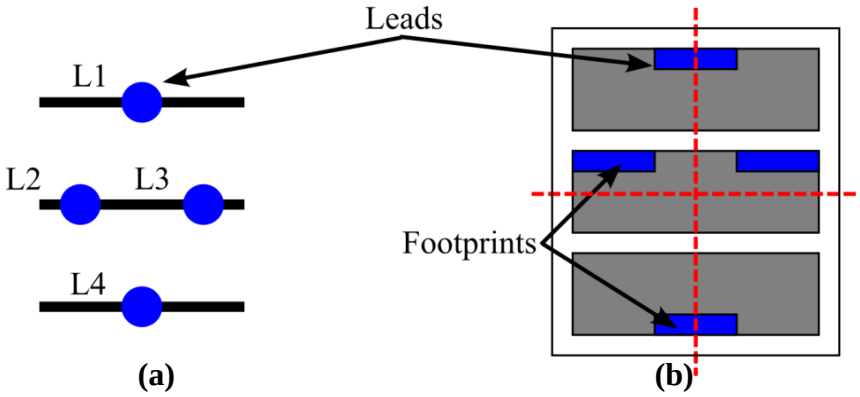


Fig. 5.14: Lead placement example. (a): Symbolic layout. (b): Geometric layout.

Leads on vertical traces are placed in a similar way, but the leads are rotated 90° and the placement methods are orthogonal analogs to the horizontal methods. This section completes the layout generation process. All trace geometries and component positions are determined and a module solution is ready for design rule checking (DRC), thermal model evaluation, and electrical parasitic model evaluation. Thousands of module solutions are generated, checked, and

evaluated during the optimization process. The next section describes how the optimization process can be improved and controlled with design symmetries and constraints. The last section in the chapter elaborates the DRC process.

5.7 Design Variable Correlations and Constraints

Design variable correlation in this context is used to represent two or more design variables which always share the same value. This means that multiple design variables are actually merged into a single variable. This reduces the amount of design variables in the overall design problem and allows for faster and more efficient optimization. Fig. 5.15 shows a layout with a correlation between the widths of traces T1 and T2. In the horizontal direction, this problem would normally have two design variables. The correlation grouping reduces this count to a single horizontal design variable.

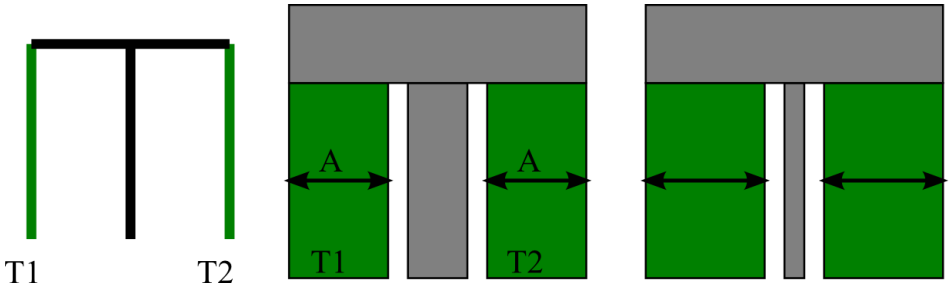


Fig. 5.15: Design variable correlation between traces T1 and T2.

Devices may also be placed in correlation groups, but the lengths of their parent traces need to be the same in order to achieve quasi-symmetrical results. This is because the devices are placed based on a percentage of the length of their parent traces.

Design variables may be limited to a fixed value or range of values by placing constraints on them. If a fixed constraint is placed on a design variable, it can no-longer vary and is removed

from the design problem. If an inequality constraint is placed on a design variable, it may only take on values on a closed interval. Constraints are bound to certain design variables through the layout line and point elements. Since each element is assigned a design variable, the constraints placed on the layout elements become associated with the correct design variable. The design variables with inequality constraints are checked for compliance during the DRC process.

5.8 Design Rule Checking

The DRC process developed in this thesis for MCPMs is similar to DRC for integrated circuits except for some extra features that aid the optimization process. An inequality constraint check and layout overflow check are executed in addition to a design rule check that resembles that of integrated circuits. The inequality constraint check ensures that the inequality constraints on design variables are met. The layout overflow check ensures that the combined width of the columns and rows equals the fixed layout width and length, respectively. These two extra checks do not meet the definition of conventional design rule checks, but play an important role in determining whether a layout meets critical design criteria. The set of conventional design rules developed for MCPMs is shown in Table 5.1, and their corresponding graphical representations are shown in Fig. 5.16.

Table 5.1: Process design rules and symbols used in Fig. 5.16.

Description	Symbol
Min. trace to trace width (gap width)	A
Min. trace width	B
Min. die to die distance	C
Min. die to trace distance	D
Min. power bondwire landing to trace distance	E
Min. signal bondwire landing to trace distance	F
Min. power bondwire landing to component distance	G
Min. signal bondwire landing to component distance	H

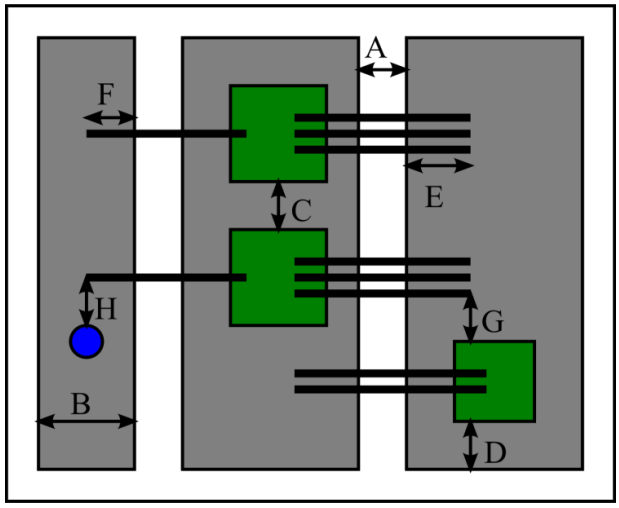


Fig. 5.16: Process design rules for MCPM.

Each of the particular design rules listed in Table 5.1 is checked by iterating through the groups of objects involved in the design rule and ensuring the geometry meets the rule. For example, the design rule “min. trace width” is checked by iterating through the trace element objects and measuring the width of each objects trace rectangle. If the width is greater than the design rule value, the trace element passes the check. If any of the trace elements does not pass,

the entire module solution is flagged as not passing. This procedure is carried out for each design rule.

At the start of optimization, many initial layout solutions do not pass the DRC. The entire first generation of solutions usually does not pass the DRC. If solutions are simply flagged as not passing, the optimizer is not provided any information as to the quality of the solutions or how close they are to passing. Some solutions may have far more DRC errors than others and some may have only single DRC errors that are extremely incorrect. If the optimizer has information which quantifies DRC error amounts, it is able to pick lower error solutions over higher error solutions. A separate error measure is given for each design rule. Most of the design rules are simple distances, thus the amount of distance the rule is violated comprises the error measure. The error measure for each rule violation is summed into a total error measure which is passed to the optimizer.

If all rules pass, the optimizer evaluates the thermal and electrical models and uses the chosen performance measures as normal. If not all of the rules pass, the optimizer instead replaces each performance measure with the same relatively large constant (as compared to the usual performance values) plus the total DRC error measure. Each performance measure holds the same value in this case, thus the equivalent of a single objective optimization is performed. This information helps guide the optimizer to DRC "clean" solutions. Multi-objective optimization seamlessly reengages once one or more DRC "clean" solutions are found.

This chapter concludes the description of the primary features which comprise the MCPM layout synthesis tool. The integration of the modeling, optimization, and layout

generation features into a single tool via a set of data structures and software flow is described in the next chapter.

Chapter 6 Software Design and Integration

6.1 Introduction

The primary software components such as the thermal model, parasitic model, optimizer, and symbolic layout system are linked together by a group of data structures and a software flow. The data structures are objects which represent physical parts used in a module and are organized into three levels of abstraction. The most general level of abstraction is the library level in which material properties and fixed dimensions for parts such as baseplates, substrates, devices, attaches, etc. are defined. These are properties of the objects which do not change on an inter-project basis. The project level governs approximately the same set of parts, but defines more specific properties such that an optimization problem may be formulated (e.g., baseplate convection coefficient, substrate width and length, etc.). Project level properties are those that do not change over the optimization process. The solution level is the most precise level in which the physical placement of devices, leads, and bondwires are represented. These are properties which change from solution to solution (the optimization results). The symbolic layout object is also a major data structure which provides a framework for this data over the entire software flow. The software flow section of this chapter describes how the data structures are used by the primary software components to move from the library level of abstraction down to a set of actual MCPM design solutions.

6.2 Software Data Structures

The primary software data structures hold information about the physical parts which comprise an MCPM. Table 6.1 shows a listing of parts found in an MCPM along with whether a

corresponding data structure for it exists at different levels of abstraction. For example, a substrate consists of two different materials (metal and dielectric) with some fixed thicknesses. The materials and thicknesses are inherent to a particular type of substrate which may be ordered or manufactured. It is likely that the same type of substrate may be used on multiple projects, thus storage of this information at a library level is advantageous. It is unlikely that the exact same width and length of substrate may be used on an inter-project basis, thus this information is instead stored at the project level in a new data structure which contains a reference to the library level object. Since the width and length of the substrate is not allowed to be modified over the optimization process, no data structure for the substrate exists at the solution level. The information regarding the substrate does not change from solution to solution in a project context.

Table 6.1: MCPM parts and levels of abstraction.

Part	Library level	Project level	Solution level
Baseplate	X	X	-
Substrate	X	X	-
Substrate Attach	X	X	-
Device Attach	X	-	-
Device	X	X	X
Bondwire	X	-	X
Lead	X	-	X

Note: "X" denotes that a part is included in a particular level of abstraction.

The library level of abstraction is analogous to a part catalog. Some parts such as the baseplate and substrate are sheets of material which are cut to particular dimensions for different projects. Therefore, only some information such as thickness of layers or material properties is

included in the library level and not project specific information. Other parts such as semiconductor devices have all of their dimensions defined and may even include a particular model of their operation. Device dimensions and properties are generally not modifiable after a device is manufactured, thus are included in the library level data structure. A data structure for devices exists at all three levels of abstraction which makes them ideal for describing the interconnection between the three differing levels. For this reason, the pseudocode for the device data structures is used to illustrate concepts at each level for the remainder of this section. The pseudocode for each data structure is shown in Appendix D. The pseudocode for the device data structure at the library level is as follows:

```
class Device:           # Library level data structure for devices
    name                # Part name of device: (String)
    dimensions          # Dimensions [width, length, thickness]: mm (List of floats)
    properties          # MaterialProperties object: (Reference)
    device_type         # Determines device type [diode or transistor]: (Integer)
    wire_landings       # List of bondwire landing objects: (List of objects)
    device_model        # Verilog-A model: (String)
```

The fields such as device_type, wire_landings, and device_model are specific to semiconductor devices, but the properties field is inherent in all library level objects. Every part type is required to have a material property definition which is used by the physical models. A **MaterialProperties** object holds the material information and is represented by the following pseudocode:

```
class MaterialProperties:
    name            # Name of material: (String)
    thermal_cond    # Thermal Conductivity: W·m^-1·K^-1 (Float)
    spec_heat_cap   # Specific Heat Capacity: J· kg^-1·K^-1 (Float)
    density         # Density: kg·m^-3 (Float)
    electrical_res  # Electrical Resistivity: ohm·m (Float)
    rel_permit      # Relative Permittivity: unitless (Float)
    rel_premeab     # Relative Permeability: unitless (Float)
```

The material properties for each part and other part specific details at the library level are entered through a GUI called the Technology Library Editor. The technology library is a database of all the technology level objects. A user may add many different types of baseplates, substrates, etc. through the Technology Library Editor which are used on an inter-project basis. The Technology Library Editor is further detailed in Chapter 7.

The data structures at the project level store information which is invariant over the optimization process, but is not usefully stored at the library level. For example, the width and length of the substrate are chosen for a project. These dimensions are fixed over the optimization process. Every solution that the optimizer generates will share the same substrate dimensions, but different projects may not share the same substrate dimensions. A substrate object at the project level is called a **SubstrateInstance**. Every object at the project level has a suffix of "**Instance**" to delineate it from the library level objects. A **SubstrateInstance** object stores the width and length of the substrate, but also contains a reference to the chosen **Substrate** object at the library level which contains material property and thickness information. As another example, the pseudocode for the **DeviceInstance** object is:

```
class DeviceInstance:
    device_tech          # Device library level object (Reference)
    attach_tech          # Attach library level object (Reference)
    attach_thickness     # Thickness of attach for this device (Float)
    heat_flow            # Heat flow generated by this device (Float)
```

A **DeviceInstance** object is created when a user selects a type of device from the technology library and pairs it with a type of device attach (solder), attach thickness, and estimated heat flow generated by the device. All of these properties are invariant over the optimization process. **DeviceInstance** objects are bound to point elements in the symbolic layout and a single **DeviceInstance** may be bound to multiple point elements. All project level objects are created through a GUI called the Project Builder. The Project Builder is where a user selects the particular devices, materials, and performance measures in which an optimization problem will be formulated. The Project Builder is described in further detail in Chapter 7.

The solution level of data structures store the most specific information about the parts used in an MCPM. The positions of devices, bondwires, and leads are all defined at this level. The implementations of the objects at the solution level carry the suffix "**Solution**". The pseudocode for the solution level device object is as follows:

```
class DeviceSolution:
    position             # Center position relative to substrate [x, y] (List of floats)
    device_instance      # DeviceInstance project level object (Reference)
    footprint_rect       # Footprint of device on substrate (Rectangle object)
```

A **DeviceSolution** object is created for each device in a layout. **LeadSolution** and **BondwireSolution** objects are created also. Every solution level object contains a reference to a project level object or library level object which contains material property or other part specific information. In chapter 5, the layout generation process creates trace rectangles and places

devices, leads, and bondwires at specific locations in a layout. The layout generation process is carried out by a method in the symbolic layout object which generates an object called **ModuleDesign** given a set of design values (the design variable vector). A **ModuleDesign** object completely defines the geometry of an MCPM and has the following pseudocode implementation:

```
class ModuleDesign:        # Stores complete MCPM design solution
    devices                # list of DeviceSolution objects
    bondwires              # list of BondwireSolution objects
    leads                  # list of LeadSolution objects
    traces                 # list of Rectangle objects (from symbolic line elements)
    substrate              # SubstrateInstance object
    substrate_attach       # SubstrateAttachInstance object
    baseplate              # BaseplateInstance object
```

The **ModuleDesign** object not only holds solution level objects but also the project level objects of the substrate, substrate attach, and baseplate. A full list of the rectangles which form the trace layout is also included. The export functions of the tool take **ModuleDesign** objects as input and output a particular type of file format e.g. Q3D or SolidWorks. The next section of the chapter describes the overall software flow from the start of the tool to the export of a solution.

6.3 Software Flow

A set of input data is required in order to run the tool and generate an optimal solution set which a user is able to browse and select best fitting solutions. The overall input and output relationship of the tool is shown in Fig. 6.1. The physical materials and dimensions of the baseplate and substrate are chosen and a symbolic layout is loaded into the tool. The materials and some of the dimensions for the substrate and baseplate parts are included in the library level data structures. Some extra data is required to form the project level objects of the baseplate and

substrate which is collected by the Project Builder. All of the input data for the tool is collected through the Technology Library Editor and the Project Builder GUIs.

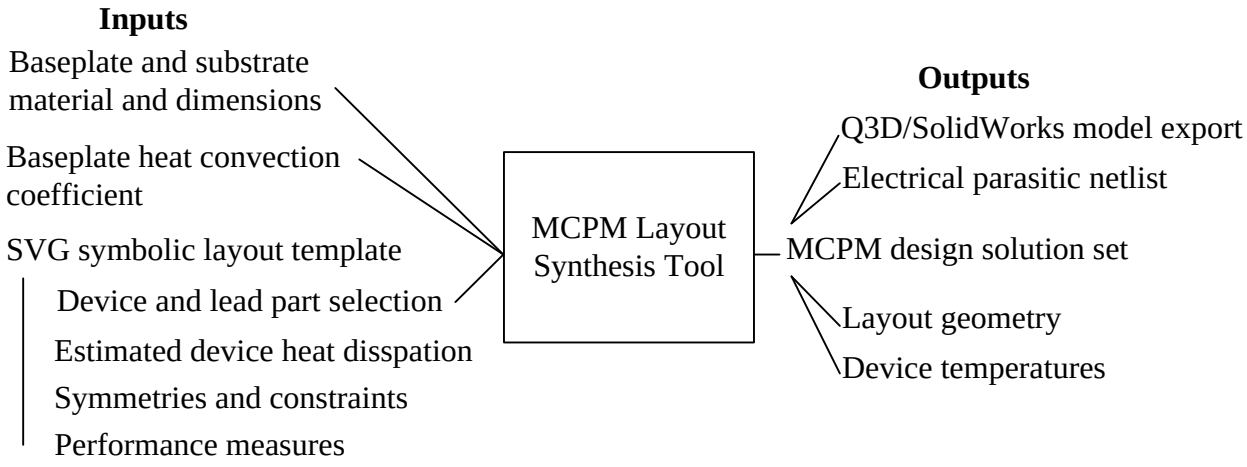


Fig. 6.1: Overall input and output of layout tool.

A symbolic layout is loaded into to the tool from an SVG file, but the raw symbolic layout contains no information about the meaning of point and line elements. Specific devices and device attaches are chosen from the technology library and paired together to form the project level device objects (**DeviceInstances**). Specific lead and bondwire types are chosen from the technology library also. A graphical rendering of the symbolic layout is shown to the user and the point and line elements in the layout are selected and bound to the chosen device, lead, and bondwire objects. A similar method is used for identifying symmetries, constraints, and performance measures. This entire process takes place in the Project Builder and the steps are reflected in the flowchart in Fig. 6.2.

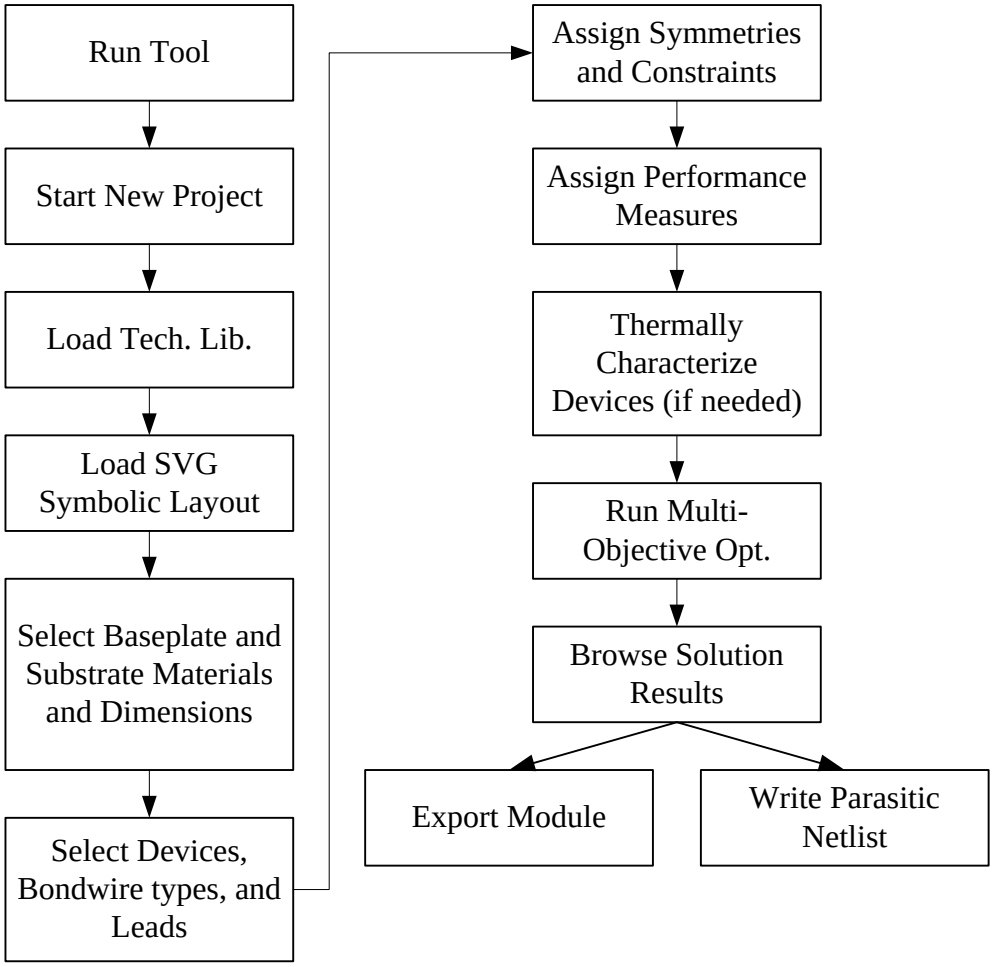


Fig. 6.2: Software flowchart.

After the identifying the parts, materials, dimensions, and performance measures of an MCPM project, a multi-objective optimization problem is formulated from the symbolic layout and a set of Pareto optimal solutions is generated. The designer/user is able to select solutions from this set and export them to other tools for more detailed analysis. These processes can be carried out in the Python scripting language, but graphical user interfaces (GUIs) reduce the effort of setting up a design problem. The GUIs involved in the tool are discussed in the next chapter.

Chapter 7 Graphical User Interfaces

7.1 Introduction

The graphical user interfaces developed for the layout synthesis tool ensure data is entered in the correct format and guide a user through the problem definition and setup process. The interfaces also allow a user to view a set of optimal layout solutions, filter them, and make a fitting selection for a problem. Three main user interfaces comprise the overall tool: the Technology Library Editor, the Project Builder, and the Solution Browser. Each of these interfaces corresponds to the three levels of abstraction relative to projects as discussed in the previous chapter. The technology library is used to create new instances of the data structures defined at the library level. The Project Builder is used to create data structures which correspond to the project level and also identify components, symmetries, constraints, and performance measures on a symbolic layout. After the optimization process is run, the Solution Browser interface is used to select layout solutions. All of the interfaces in the tool are written using Python and the PySide Qt cross-platform GUI toolkit [30], [31].

7.2 Technology Library Editor

The Technology Library Editor (TLE) is used to create entries for MCPM parts in the technology library. The TLE is comprised of a series of pages which form a wizard type interface. A separate page exists for each of the 7 categories of MCPM parts. A user is able to navigate forward or backward through any of the pages at any time. The TLE is used to create new part entries or edit existing parts. The part categories and their relation to an MCPM are

illustrated in Fig. 7.1. The TLE is run independently from the rest of the tool and used to build a library of information generally before a project is started.

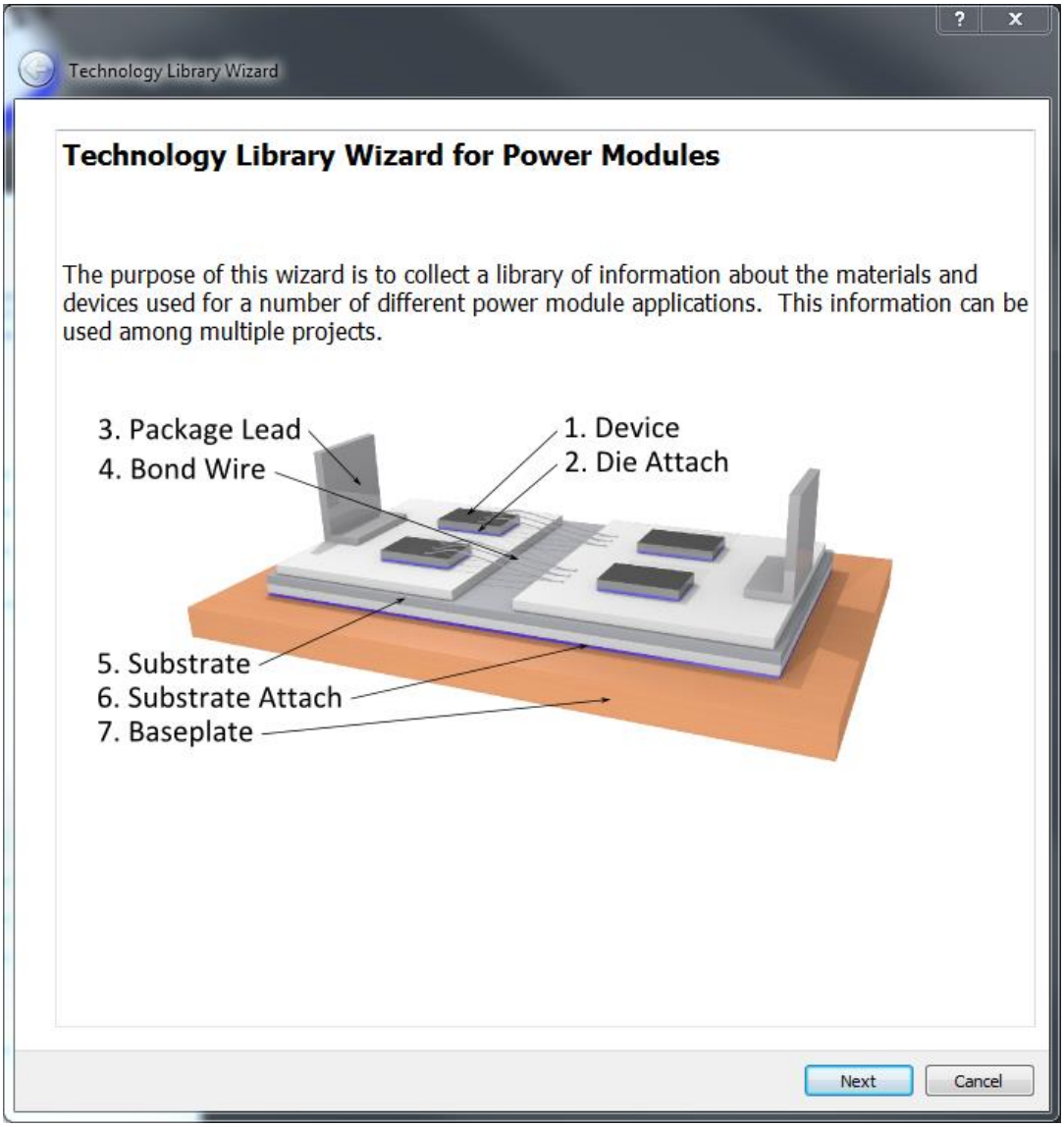


Fig. 7.1: Start-up screen for technology library editor/wizard (the above software screenshot is intended only to give the reader a sample view of the software discussed).

The part entry pages or forms each have a similar form to one another. At the top of the page is a list box which contains any pre-existing part instances in the library. A group of text fields or other input fields comprise the bulk of the page. A save button and part name field

located near the bottom page are used to save the part information to the library. Fig. 7.2 shows the Device page from the TLE. All of the TLE part pages are shown in Appendix E (excluding the Device page).

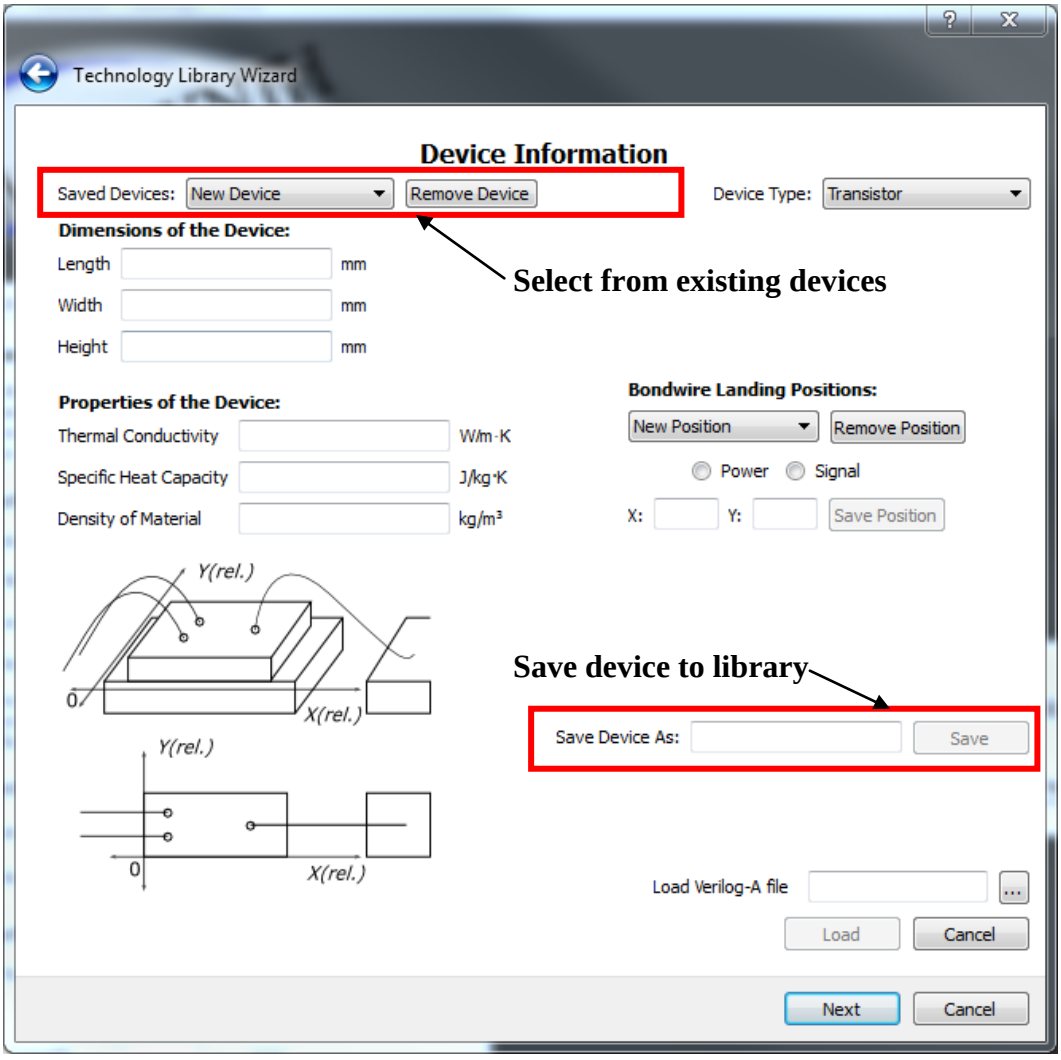


Fig. 7.2: Device page for technology library editor (the above software screenshot is intended only to give the reader a sample view of the software discussed).

In order to create a new part, the fields of the page are filled in by the user with desired values, a name is given to the part, and the save button is clicked. The data fields are validated and if any of the data doesn't meet a specified format the user is notified. Each part is saved to its

own file on disk. The technology library is composed of a set of directories with a corresponding directory for each part type. A collection of part files in each directory. The part files are "pickled" versions of the Python library level data structures. Python allows instances of data structures to be "pickled" or represented in a format which may be written to disk. The object and its data can be reloaded into program memory by reading the pickle file on disk and passing the data through Python's load pickle method. This allows objects to be saved and loaded from disk in an easy and efficient manner.

Parts may also be edited or modified using the drop down list box located near the top of each part page. Once a specific part is selected from the list box, the data fields for the page are automatically loaded with the part's information. The modified information for the part is saved if the save button is clicked.

7.3 Project Builder

The Project Builder interface is used to guide a user through the setup procedure for executing a layout synthesis and optimization process. The Project Builder is composed of 6 different pages which break down the setup procedure into simple tasks. The pages can be accessed in any order, but are designed to flow from materials and component selection to design problem configuration. All of the fields on each page are validated when the "Run Optimization" button is clicked on the Results page. Each page is composed of a navigation and data entry component of the left side and a large sometimes interactive figure on the right. The first of these pages is the Module Stack page which is shown in Fig. 7.3.

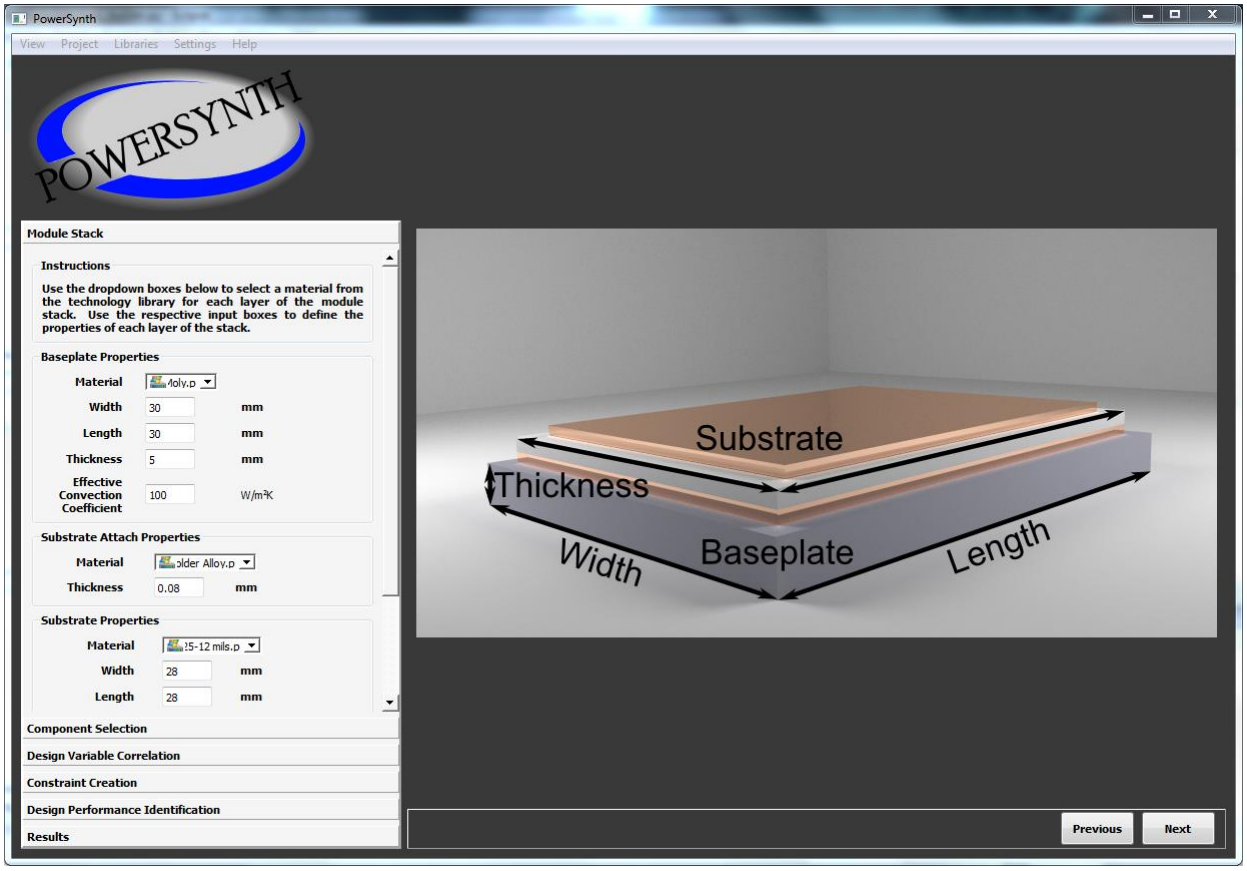


Fig. 7.3: Project Builder interface with the Module Stack page selected (the above software screenshot is intended only to give the reader a sample view of the software discussed).

The primary purpose of the Module Stack page is to collect information about the baseplate, substrate, and overall system properties. The baseplate, substrate, and substrate attach materials are selected from a drop down box which contains a list of parts/materials loaded from the technology library. The rest of the fields on the page are numeric text fields. A close-up of these text fields for the Module Stack page is shown in Fig. 7.4.

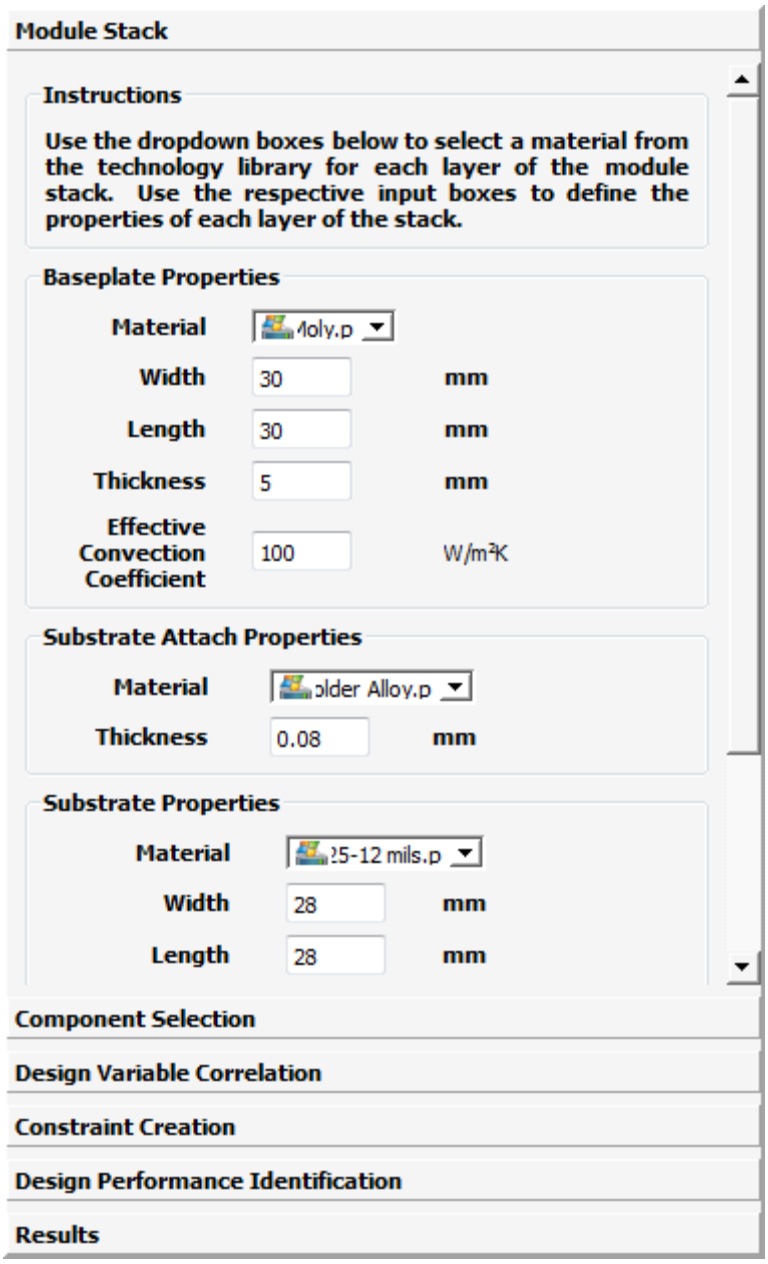


Fig. 7.4: Navigation and data entry component (the above software screenshot is intended only to give the reader a sample view of the software discussed).

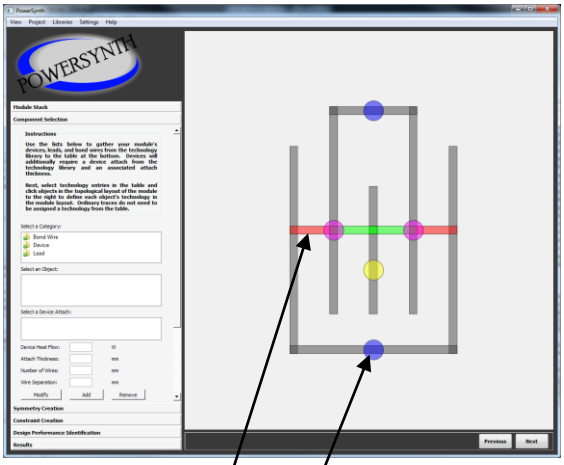
Fig. 7.4 is the navigation and data entry component which resides on the left side of every Project Builder page. The navigation component is shown to have the Module Stack page selected in Fig. 7.4. Each page contains its own set of data fields which reside in the navigation

component. A set of instructions is included in the navigation component to help inform the user how the data fields work for each page.

The Component Selection, Design Variable Correlation, Constraint Creation, and Design Performance Identification pages all contain an interactive symbolic layout figure to the right of the navigation component. The Component Selection page is shown in Fig. 7.5(a) and a close-up of the navigation component for the page is shown in Fig. 7.5(b). A list of devices, leads, and bondwires are selected and added to a list which is displayed near the bottom of the navigation component. The devices, lead, and bondwires in the list are able to be selected and "painted" onto the interactive symbolic layout figure. A unique color is given to each list entry which aids in identifying which component is bound to the symbolic layout components. This is the process by which the library and project level data structures become bound to the symbolic layout objects.

A similar process is carried out for the Design Variable Correlation, Constraint Creation, and Design Performance Identification pages. Items are created, added to a list, selected, and bound to symbolic layout elements. Images of these pages can be found in Appendix F.

(a)



(b)

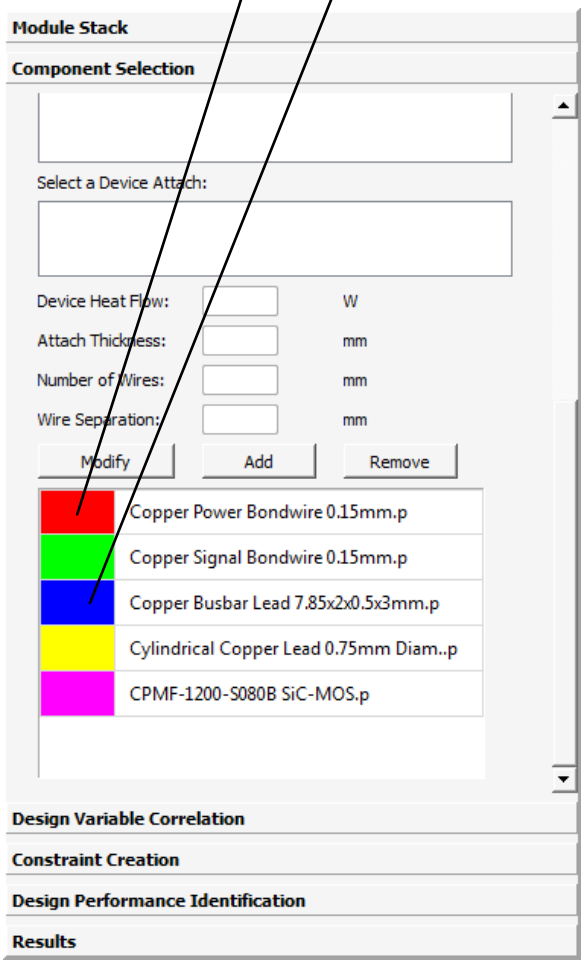


Fig. 7.5: Screen capture of Component Selection page (the above software screenshot is intended only to give the reader a sample view of the software discussed). (a): Overall page with interactive symbolic layout. (b): Close-up of list of selected components.

The last page of the Project Builder is the Results page which serves two functions. The first is to select the number of generations in which to run the optimization algorithm and initiate the optimization process. The second is to view saved solution results in greater detail. Solutions can be selected and saved from the Solution Browser. The Solution Browser is automatically displayed after the optimization process is completed and is discussed in further detail in the next section.

Fig. 7.6: Results page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

The process design rules for a project can also be modified by the Process Design Rules Editor dialog shown in Fig. 7.7. This dialog is accessed from the menu bar at the top of the Project Builder window under Project->Design Rules Editor.

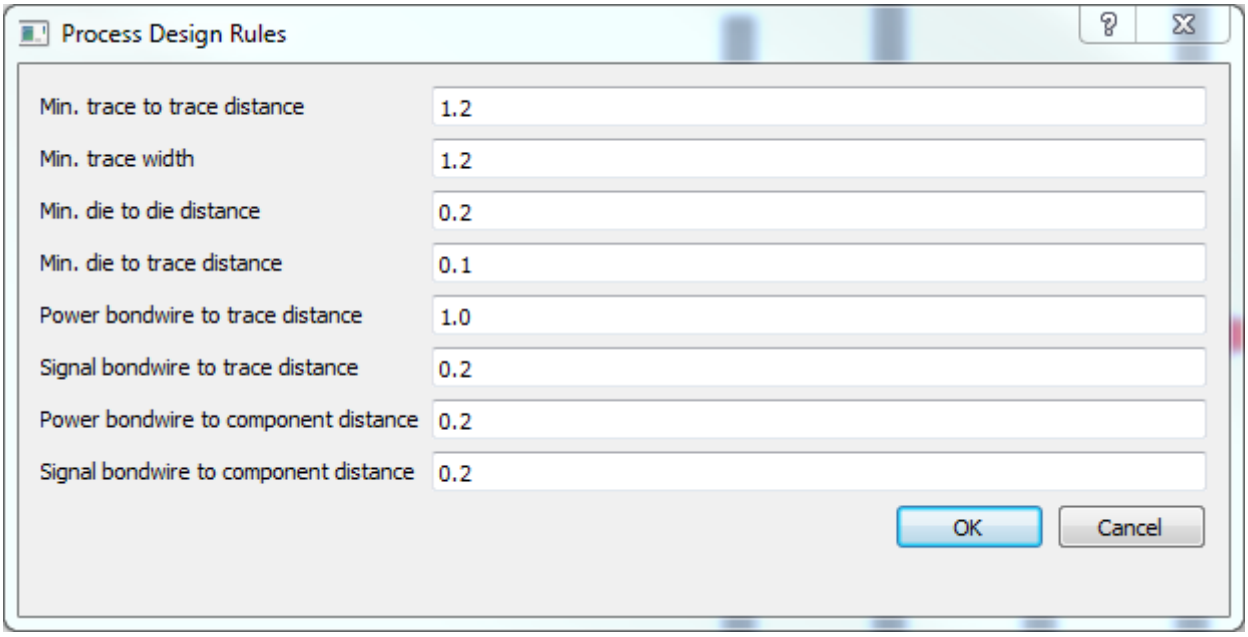


Fig. 7.7: Screen capture of Process Design Rules Editor dialog (the above software screenshot is intended only to give the reader a sample view of the software discussed).

7.4 Solution Browser

The Solution Browser is used to filter, select, and save optimal layout solutions generated by the tool. Fig. 7.8 shows the overall layout of the window. It is composed of three main components: the plot filter, the solutions plot, and a layout preview of the currently selected solution. A button at the bottom left of the window is used to save a selected solution to the Project Builder Results page.

The solutions plot component is an interactive plot of the Pareto optimal solutions found by the optimizer. 2D or 3D plotting capabilities are available depending on the number of performance measures chosen to be displayed in the plot filter. The plot shown in Fig. 7.8 is a 3D plot based on 3 performance measures shown on the plot axes. The mapping of which axis receives which performance measure is chosen in the filter. Each point in a plot represents a

unique layout solution to the multi-objective optimization problem cast in the Project Builder. If a solution point is clicked, the layout preview is updated to show the geometric layout of the selected solution. A plot can be translated, scaled, or rotated using the toolbar at the bottom of the plot or by clicking and dragging the plot area. The solutions plot and layout preview are both rendered using the matplotlib Python graphing library [32].

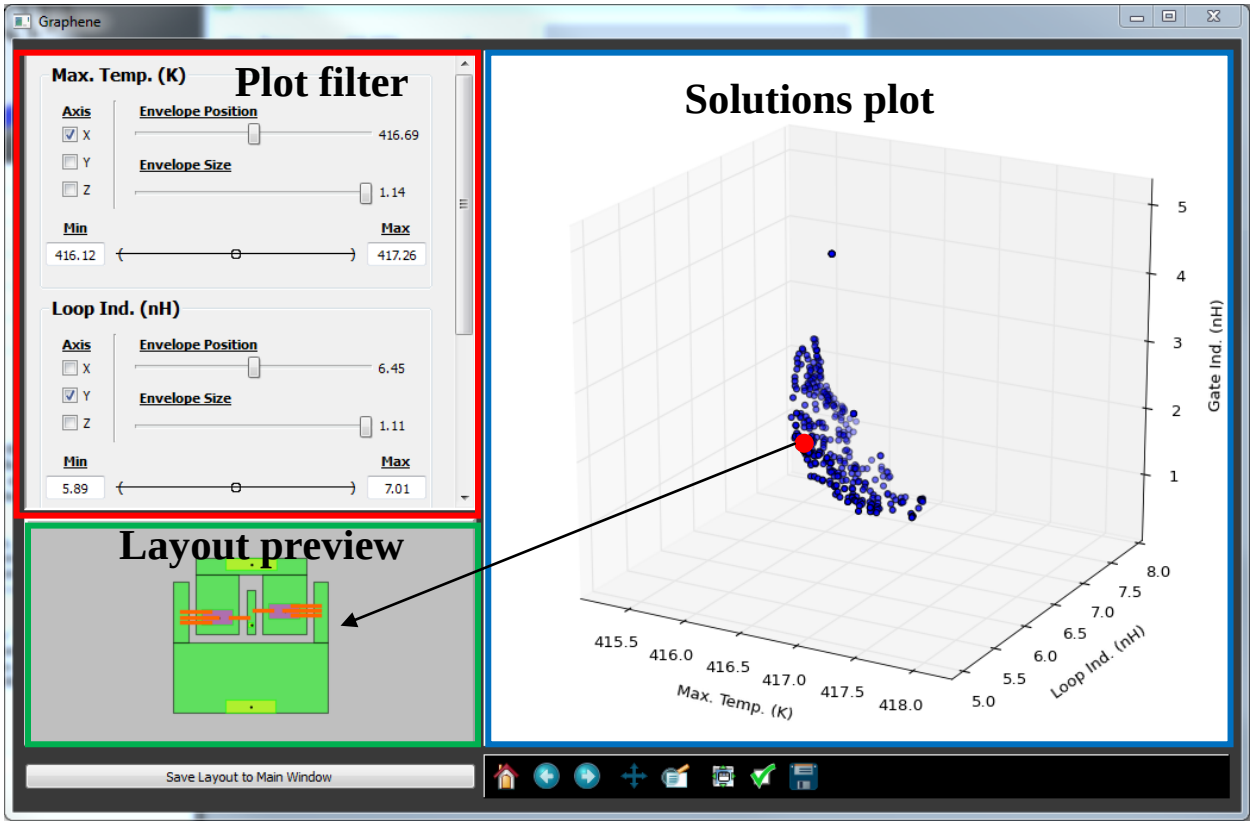


Fig. 7.8: Solution Browser window (the above software screenshot is intended only to give the reader a sample view of the software discussed).

The number of performance measures or objectives in the optimization problem may exceed 3, and in this case all objectives may not be able to appear on the plot's axes simultaneously. Under these circumstances, a visually complex set of solution points may appear in the 3D or 2D plot. The sliders in the plot filter component are used to alleviate complexity and

aid the user in searching through the higher dimensional solution space. An envelope position and envelope size slider is provided for each objective. If a solution's objective values fall within the corresponding envelope, the solution is plotted. The envelope is an interval whose midpoint is determined by the envelope position slider and range is determined by the envelope size slider. Generally, as an envelope's size is decreased the total number of solutions plotted decreases. A user can interactively increase or decrease the position slider on an objective with no axis assigned and watch a "family of curves" evolve in the plot. The envelopes can also be used to simply filter out unwanted regions of solutions.

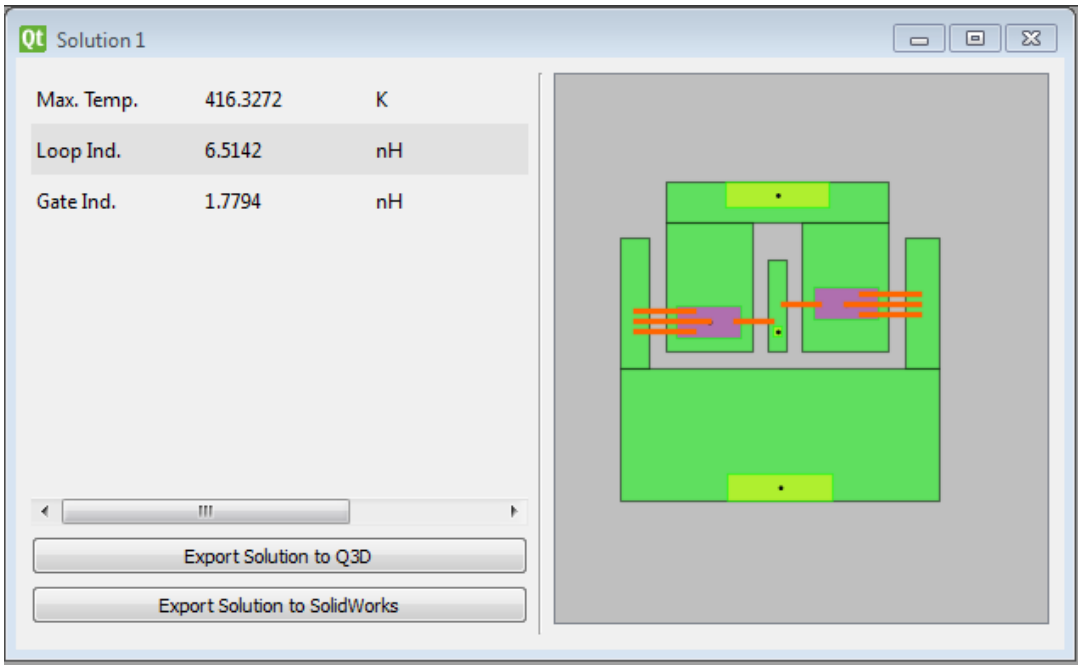


Fig. 7.9: Saved solution sub-window within the Project Builder (the above software screenshot is intended only to give the reader a sample view of the software discussed).

Once a user has selected and saved a solution to the Project Builder, it is displayed on the right side of the Results page in a sub-window. From this window, a user can view the layout

geometry in more detail, view all of the objective function values, and export the design to either Q3D or SolidWorks.

Chapter 8 Results

8.1 Design Example (P-Cell N-Cell Half-Bridge Inverter Module)

This section investigates the design of a half-bridge inverter module with separate anti-parallel diodes. The addition of anti-parallel diodes outside of the MOSFET die allows for an opportunity to reduce stray inductance in the package using a technique called P-cell N-cell layout [33]. Conventionally, the anti-parallel diodes are either packaged within the MOSFETS or placed closely in parallel with them as shown in Fig. 8.1(a). The P-cell N-cell layout technique reduces the stray inductance of the main current commutation pathways in a module which cause voltage overshoot and current ringing during switch turn-off and turn-on, respectively. This is done by pairing the upper anti-parallel diodes with the lower MOSFETS and vice-versa, as shown in Fig. 8.1(b). The closer these devices are paired together, the lower the parasitic inductance between the devices. This lower inductance reduces the inductive voltage spikes seen across a MOSFET while undergoing a turn-off event and current overshoot during turn-on [33].

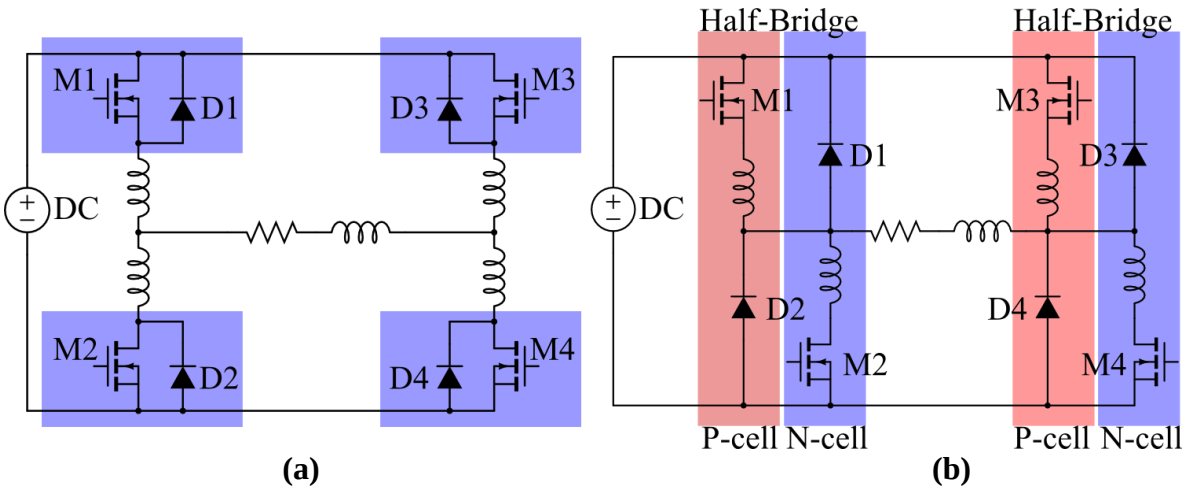


Fig. 8.1: Schematic of full-bridge inverter with parasitic inductance. (a): Diodes internal to MOSFETS. (b): P-cell N-cell paring.

8.2 Optimization Objectives

The optimization objectives for this particular example include the minimization of 5 main criteria: the loop inductance in the pathway from the positive to negative terminal, gate to source loop inductance of upper and lower switching positions, maximum device temperature, and the standard deviation of the device temperatures. These 5 criteria represent the performance measures used for multi-objective optimization.

By minimizing the total loop inductance in the layout from the positive to negative terminals, the inductance of both current commutation pathways are simultaneously minimized. This reduces the number of performance measures by one, instead of measuring both pathways. Two separate performance measures are created for the gate to source loop inductances. The minimization of the gate loop inductances helps reduce ringing, but they have been shown to be not as important in terms of ringing and switching loss energy as the main loop inductance [6].

Reducing maximum device temperature is helpful in terms increasing the reliability and longevity of the module. Generally, the lower the operating temperature of the module, the lower thermal expansion and thus stress between the devices and packaging [3]. Additionally, the standard deviation of the device temperatures is to be minimized to help reduce the difference in temperature between the parallel devices. This helps the devices all operate within the same temperature range and have similar on-state resistances, thus helping balance the currents through each device.

8.3 Devices and Heat Flow

There are a total of 6 MOSFETs and 6 anti-parallel diodes included in the design of this module, where 3 MOSFETs are used in each of the upper and lower switching positions. This module is a half-bridge, so only a single inverter phase-leg is implemented. The MOSFETs used in the design are CREE CPMF-1200-S160B and the diodes are CREE CPW4-1200S020B. The module is assumed to be operating at an ambient temperature of 300 K and at a switching frequency of 50 kHz. The bottom face of the baseplate is given an effective convection coefficient of 100 $W \cdot m^{-2} \cdot K^{-1}$. Each MOSFET is estimated to dissipate 4 W of heat flow and each diode 2 W of heat flow. The baseplate and substrate dimensions are constrained to a fixed size. The positioning of the devices and trace sizing are allowed to be modified during the optimization process.

8.4 Symbolic Layouts

Two different symbolic layout topologies are experimented with in the optimization process. Both symbolic layouts attempt to use the P-cell N-cell layout technique by pairing the

MOSFETs closer to their corresponding diodes. The first symbolic layout, shown in Fig. 8.2(a), is based on a design provided by a layout design engineer, Atanu Dutta. An optimized geometric layout solution is shown to the left in Fig. 8.2(b).

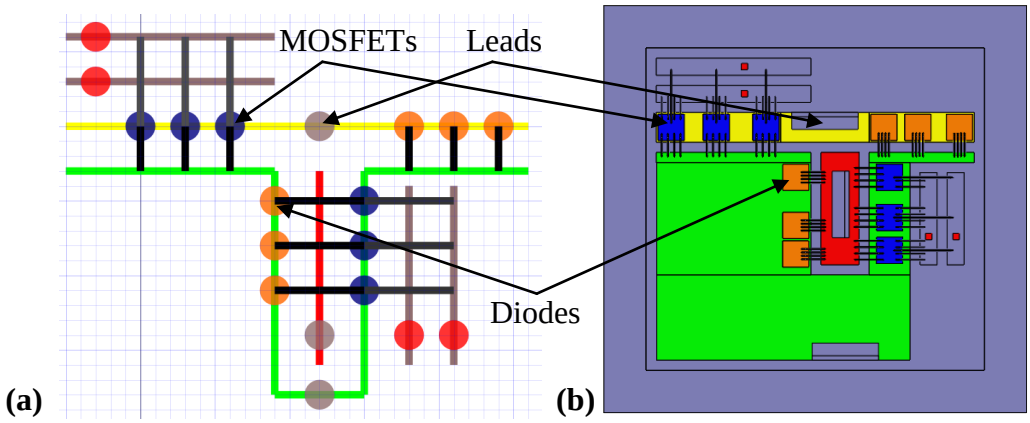


Fig. 8.2: Symbolic layout A. (a): Stick figure diagram. (b): A geometric solution.

The second layout, shown in Fig. 8.3(a), is a modified topology where the upper MOSFETS and trace has been folded downward and the upper diodes are folded inward. This is done to achieve a more compact layout design. This design may also scale better if the number of paralleled devices in the module is increased because of its more rectangular design. A geometric layout solution of this type is shown in Fig. 8.3(b).

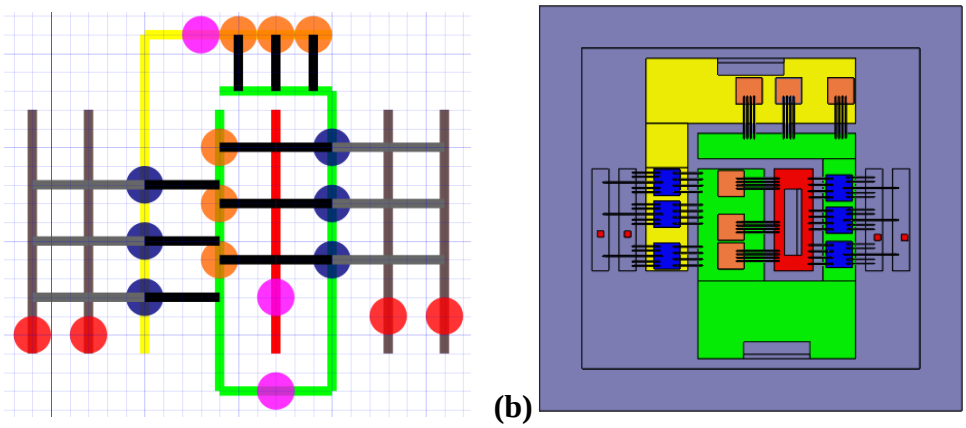


Fig. 8.3: Symbolic layout B. (a): Stick figure diagram. (b): A geometric solution.

8.5 Layout Results

The resulting Pareto front after running the NSGA II multi-objective optimization algorithm with symbolic layout A is shown in Fig. 8.4. A red curve is overlaid on Fig. 8.4 to indicate an approximate Pareto front. This curve relates the trade-off between the main loop inductance of the module and the maximum temperature in the module. The other solutions which do not reside on the indicated front exist because of other performance measure criteria which do not have an axis allocated. Only 3 performance measures can be graphed at once due to restrictions of plotting at maximum 3 variables at once on a graph.

The spread of the maximum temperature in the solutions is only about 3 K and about 2 nH for the loop inductance. A good trade-off between the two objectives is around 458 K and 4.0 nH. This is the area from which the layout solution shown in Fig. 8.2(b) is selected. A designer would generally make a decision on a specific solution based on the requirements of the system or what is considered the best solution for the system.

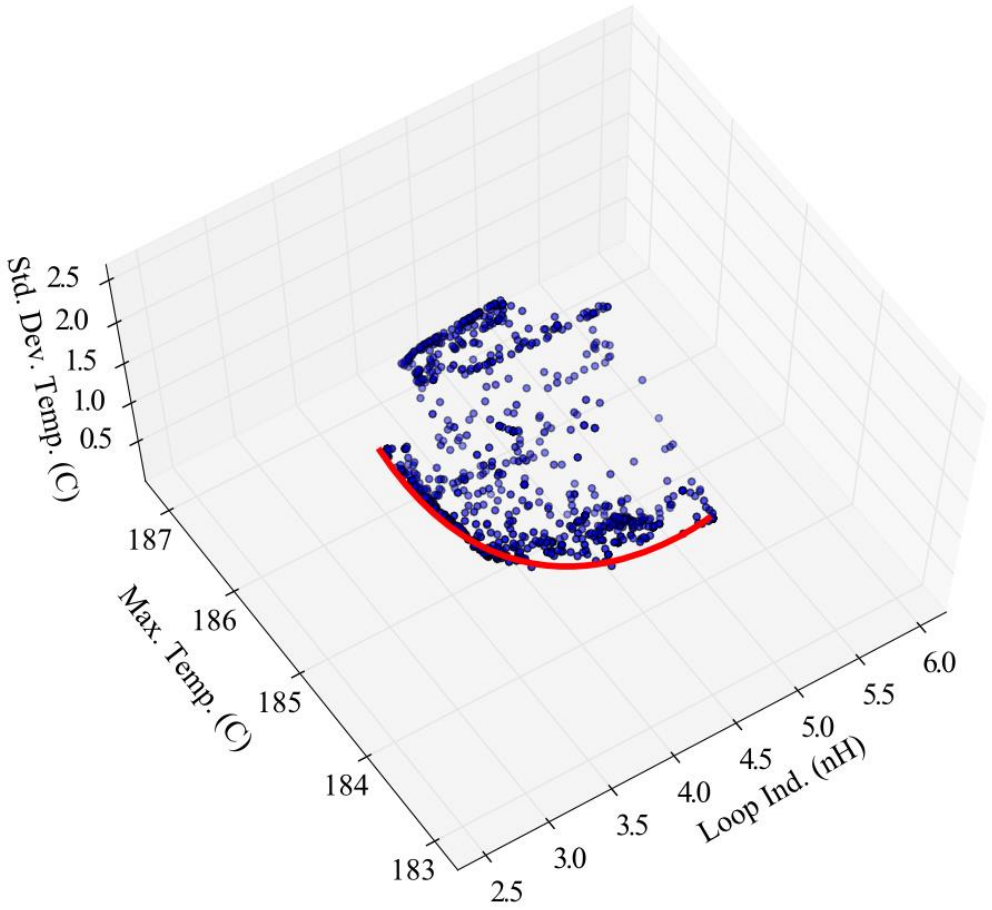


Fig. 8.4: Pareto front of symbolic layout A.

The Pareto optimal solutions for symbolic layout B are shown in Fig. 8.5. The front formed from this layout configuration is shown in 2 dimensions for simplicity and gives the trade-off between maximum temperature and loop inductance. A Pareto front had begun to form on the red curve, but some new optimal solutions formed underneath this curve. If the optimization procedure had run longer, a newer front may have formed in the location that the red circle indicates in Fig. 8.5. The geometric layout shown in Fig. 8.3(b) is chosen from this circled region. This region has the benefit of having low loop inductance and low device maximum temperature. In a later section, the two optimized layout solutions are compared with two human engineered designs.

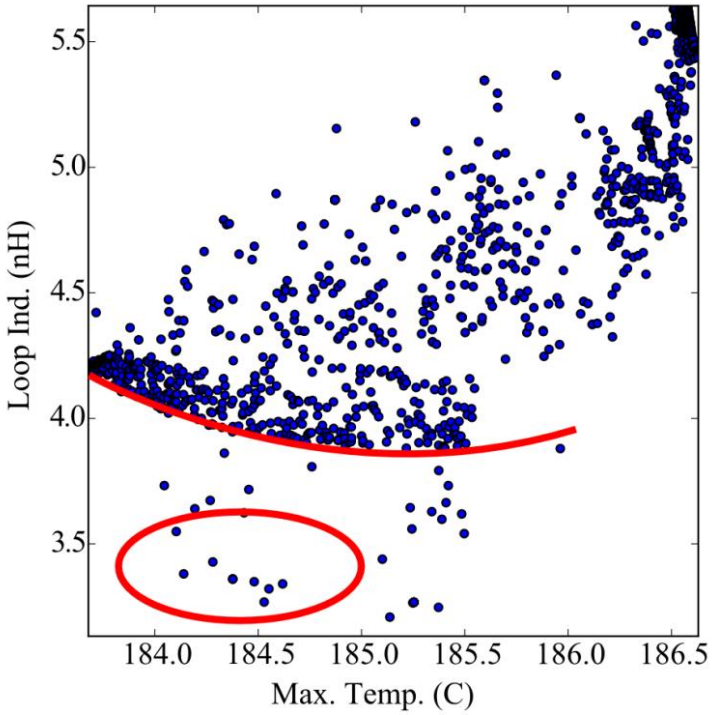


Fig. 8.5: Pareto front of symbolic layout B.

An interesting phenomenon occurred in the relation between the standard deviation of the device temperatures and the maximum temperature in both layout types. The two quantities are approximately linearly related as seen in Figs. 8.6(a) and 8.6(b). This probably has to do with way the temperature distributions around the devices are distributed. When devices are more spread out, their interacting temperature distributions are in less steeply changing regions yielding less deviation in temperatures in the module. If the devices are more closely packed together, the steeply changing portions of each device's distribution is interacting with its neighbors, thus creating unbalanced and deviated temperatures among the group.

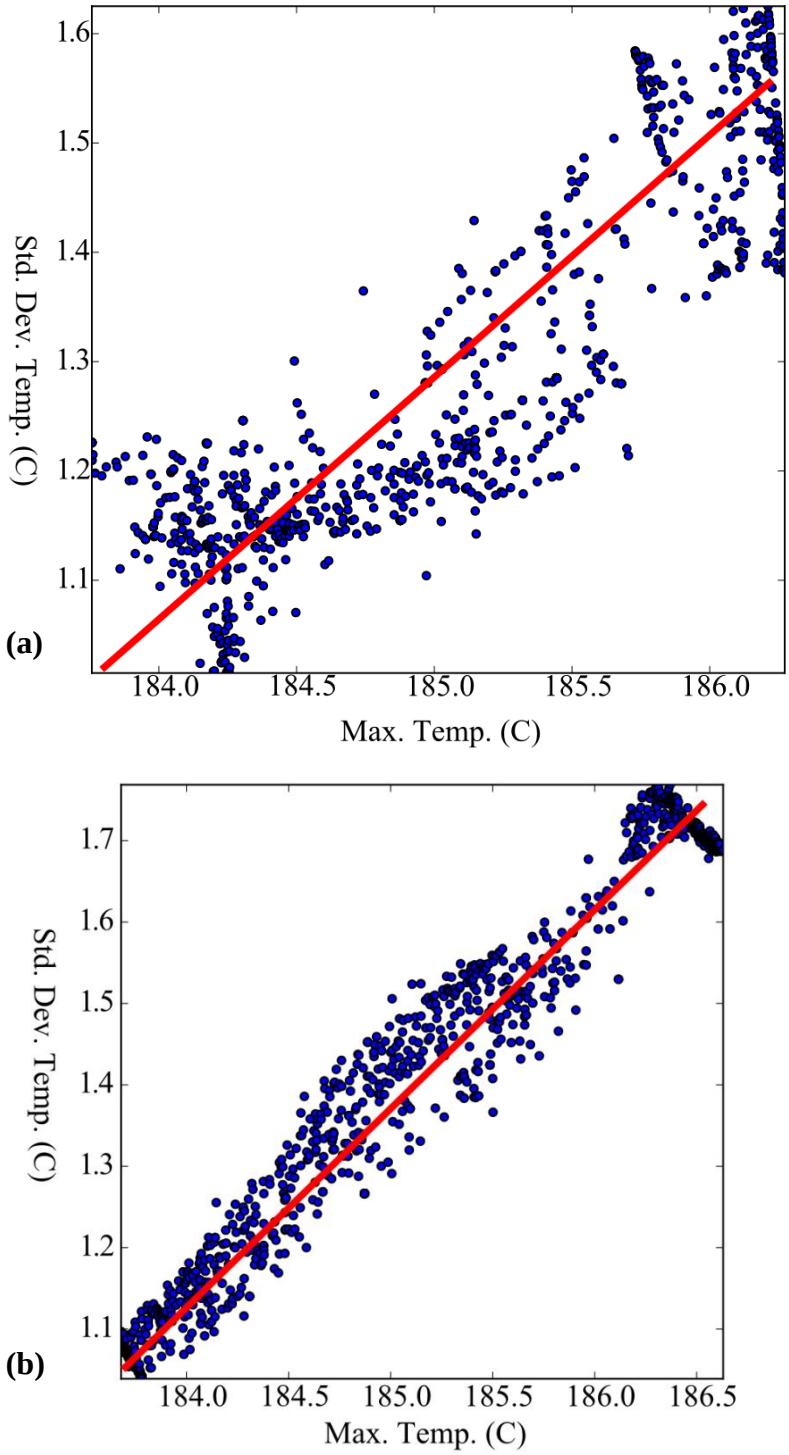


Fig. 8.6: Linear relation between max. and std. dev. of device temperatures. (a): Symbolic layout A. (b): Symbolic layout B.

The multi-objective optimization process for symbolic layout A and B using NSGA II running for 3000 generations took 576 s and 495 s, respectively. The processes were single-threaded and ran on an Intel Core i7 870 CPU clocked at 2.93 GHz. The number of generations used for evaluation is approximately linear to the run time.

8.6 Human Design Comparison and Analysis

Fig. 8.7(a) and Fig. 8.7(b) show two layouts designed by an engineer, Atanu Dutta. The first layout is a conventional layout which does not consider P-cell N-cell pairing of diodes and MOSFETS. The second is the P-cell N-cell aware design. The conventional layout also resides on a larger substrate and baseplate than the P-cell N-cell. The previous two layouts generated by the tool are designed on the same size substrate and baseplate as the P-cell N-cell layout shown in Fig. 8.7(b). All four of the layouts considered in this section were constructed in Q3D for parasitic analysis and SolidWorks for thermal analysis [11], [20]. Table 8.1 summarizes these results.

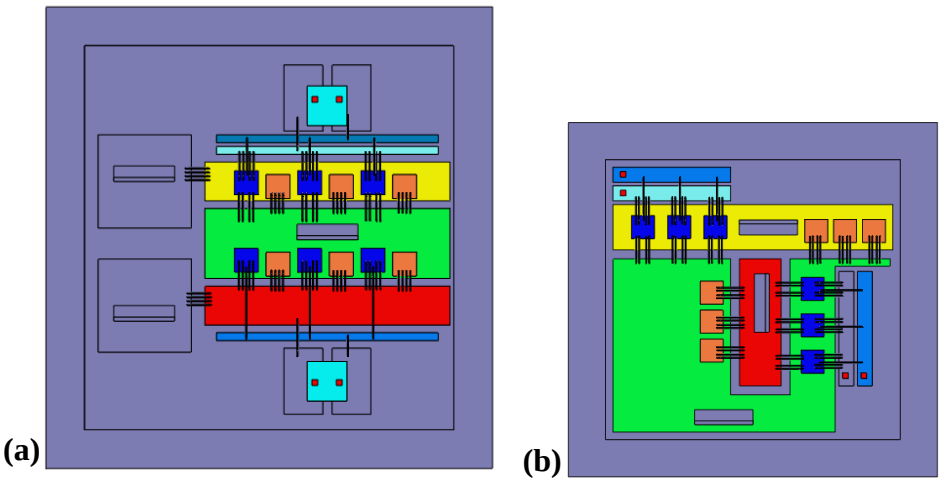


Fig. 8.7: Human Designed Layouts. (a): Conventional Layout. (b): PN Layout.

Table 8.1: Layout Performance Results.

Layout Name	Loop Ind. (nH)	Lower Gate Loop Ind. (nH)	Upper Gate Loop Ind. (nH)	Max. Temp. (°C)	MOSFET Std. Dev. Temp. (°C)
Conventional	7.45	9.37	8.08	139	0.713
PN	1.72	3.41	3.62	185	0.362
A	1.59	3.00	3.14	184	0.298
B	3.46	2.99	3.105	184	0.310

The layout with the lowest overall loop inductance and lowest deviation of MOSFET temperatures is layout A, as seen in Fig. 8.2(b). The P-cell N-cell layout configuration significantly reduces the loop inductance over the conventional layout style. The Conventional layout and module is somewhat larger in size as compared to the other 3 modules which accounts for its reduced maximum temperature. The standard deviation of its MOSFET temperatures is significantly higher than the other 3 layouts. This is most likely due to its asymmetric and interleaved grouping of diodes and MOSFETs. This makes the Conventional layout in this particular example one of the least fit designs of the group.

Layout A carries the same stick-figure design as the human designed PN layout. Some major differences between the layout A and PN are the decrease in bondwire length and increase of trace width where bondwire length is unaffected both leading to a decrease of inductance in layout A.

The “folded” layout B stands as an example for ease of design exploration. It comes close in performance to the A and PN layouts, but allows for more compact usage of module space. If

the number of paralleled die was allowed to change in the module, layout B may outperform the A and PN symbolic layouts.

Chapter 9 Conclusions and Future Work

9.1 Summary

The design and implementation of an MCPM layout synthesis tool has been presented which integrates thermal and electrical modeling techniques with a symbolic layout system and multi-objective optimizer. The tool provides a GUI in order to streamline the entire design process. The thermal modeling techniques require a characterization step for each type of device in an MCPM design. This characterization step has been fully automated such that a user is not burdened with building and solving finite element thermal models. The electrical parasitic extraction system is now capable of handling any layout scenario involving Manhattan style routing and provides accurate inductance and capacitance estimation under a wide range of layout designs. A symbolic layout system capable of representing complex MCPM layouts has been successfully integrated with a multi-objective optimization algorithm (NSGA-II). The two systems are able to work in conjunction to produce a set of approximately Pareto optimal solutions to many different design problems. A set of GUIs have been presented which allow a user to build a part and material library, develop an MCPM layout project, and explore a set of layout solutions. A user has the capability of exporting entire MCPM designs quickly for further performance verification or manufacture.

9.2 Conclusions

The field of MCPM design is rapidly evolving with the introduction of new techniques such as bondwire free modules and double sided cooling [1], [3]. It is important to ensure the MCPM design and synthesis tool is built with a modular framework in order to meet the

demands of new techniques. Physical models which support new techniques should be easy to integrate into the tool.

Including extensive export functionality for the software greatly increases productivity for MCPM design development and physical (thermal, electrical, etc.) model development. Linking the output of the synthesis tool to 3D modeling tools such as SolidWorks provides the ability to output a design in a number of standard formats and allows editing of layouts.

In order to achieve layout synthesis, a method of discretizing the layout search space is critical. The method of discretization in this thesis is the symbolic layout and its corresponding matrix data structure. A lower level optimization process can produce geometric solutions based on this symbolic representation. Reducing systems to a set of symbolic components which exist in a discrete space is critical to automating the design of systems.

A well-organized software development framework is integral to building a functioning complex tool. A development team is able to work more efficiently if the software is broken into modular components with well documented interface specifications.

9.3 Future Work and Recommendations

The current version of the tool uses premade symbolic layouts loaded from an SVG file. The automatic generation of symbolic layouts would allow the tool to achieve full layout synthesis. There may be a number of ways of automatically generated symbolic layouts. An experimental module was written for the tool which generates symbolic layouts in a standard form based on circuit schematics, although this module only supported half-bridge topologies. The down side to this approach is that a new algorithm needs to be written to handle new types

of circuit topologies. A more general method could be created by using a meta-heuristic algorithm such as genetic algorithms, simulated annealing or particle swarm optimization which can search a fixed size symbolic layout space for solutions which match a particular circuit schematic. This process may produce a number of viable symbolic layouts which would be passed to the current multi-objective optimization process.

The thermal model in the current tool does not support temperature estimation over variation in baseplate and substrate size. Improving the thermal model for this type of support is possible, but would require some research. If supported, a larger delta of trade-off temperatures versus loop inductance would be possible allowing for a very useful metric for sizing the overall module. Support for double-sided cooling and bondwire free modules would allow state-of-the-art MCPMs to be synthesized with the tool. Adding a thermal capacitance extraction step would also allow a transient thermal model to be extracted for modules. Some research into this type of extraction can be found in [34], but may not be good for optimization problems because of run times on the order of seconds for single modules.

Including a greater selection of layout components such as discrete resistors, inductors, and capacitors would allow PowerSynth to handle a wider variety of MCPM layouts. PowerSynth could also potentially handle gate driver die directly integrated onto the substrate.

The electrical parasitic extraction system provides accurate inductance and capacitance estimates, but is lacking in bondwire resistance estimation at high frequencies. This is probably due to proximity effects between the wires. Approximating these effects may greatly improve the accuracy of the resistance model. Considering mutual inductance between traces and bondwires may also be beneficial for minimizing cross-talk between power paths and gate loops. The tool is

able to produce a parasitic extraction SPICE netlist of modules, but modules with multiple devices take very long amounts of time to solve or fail to converge [15]. Model order reduction techniques or methods of lumping parasitic elements together may help improve solution time and convergence. The electromagnetic interference (EMI) of a module may also be approximated by the electrical parasitics within a layout. Providing a performance measure/objective for EMI would help designers catch EMI problems early and improve the robustness of their MCPM systems.

The electrical, thermal, and symbolic layout systems in PowerSynth are constructed around rectangular shaped traces. These three systems would need to be reworked completely in order to support more general trace shapes. Research is required to examine the benefits of support for non-rectangular trace shapes.

The NSGA-II algorithm is used to conduct the multi-objective optimization process in the current tool, but other multi-objective algorithms remain to be tried. The multi-objective particle swarm optimization (MOPSO) algorithm may be a good candidate for this process [35].

REFERENCES

[1] F. C. Lee, J. D. van Wyk, D. Boroyevich, Guo-Quan Lu, Zhenxian Liang, and P. Barbosa, "Technology trends toward a system-in-a-module in power electronics," *IEEE Circuits and Systems Magazine*, vol. 2, no. 4, pp. 4– 22, Fourth Quarter 2002.

[2] J. M. Homberger, S. D. Mounce, R. M. Schupbach, A. B. Lostetter, and H. A. Mantooth, "High-temperature silicon carbide (SiC) power switches in multichip power module (MCPM) applications," in *Industry Applications Conference, 2005. Fourtieth IAS Annual Meeting. Conference Record of the 2005*, 2005, vol. 1, pp. 393– 398 Vol. 1.

[3] H. Zhang, S. S. Ang, H. A. Mantooth, and S. Krishnamurthy, "A high temperature, double-sided cooling SiC power electronics module," in *2013 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2013, pp. 2877–2883.

[4] P. Ning, F. Wang, and K. D. T. Ngo, "Automatic layout design for power module," *IEEE Transactions on Power Electronics*, vol. 28, no. 1, pp. 481–487, 2013.

[5] N. Hingora, X. Liu, Y. Feng, B. McPherson, and A. Mantooth, "Power-CAD: A novel methodology for design, analysis and optimization of Power Electronic Module layouts," in *Energy Conversion Congress and Exposition (ECCE), 2010 IEEE*, pp. 2692–2699.

[6] Z. Chen, D. Boroyevich, and R. Burgos, "Experimental parametric study of the parasitic inductance influence on MOSFET switching characteristics," in *Power Electronics Conference (IPEC), 2010 International*, 2010, pp. 164–169.

[7] 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, no. 6, pp. 837–846, 2012.

[8] Z. Gong, "Thermal and Electrical Parasitic Modeling for Multi-Chip Power Module Layout Synthesis," M.S. thesis, EE, Univ. of Arkansas, Fayetteville, AR, 2012.

[9] 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 *2013 IEEE 14th Workshop on Control and Modeling for Power Electronics (COMPEL)*, 2013, pp. 1–4.

[10] J. N. Reddy, *An Introduction to the Finite Element Method*. McGraw-Hill Education, 2006.

[11] *SolidWorks*. Dassault Systèmes, 2012. Available: http://www.solidworks.com/

[12] *ANSYS Workbench Platform*. ANSYS, 2010. Available: http://www.ansys.com/

[13] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[14] *Elmer: Open Source Finite Element Software for Multiphysical Problems*. CSC IT Center for Science, 2013. Available: http://www.csc.fi/english/pages/elmer/

[15] P. Tucker, "SPICE Netlist Generation for Electrical Parasitic Modeling of Multi-Chip Power Module Designs," B.S. Honors thesis, EE, Univ. of Arkansas, Fayetteville, AR, 2013.

[16] *NetworkX*. Los Alamos National Laboratory, 2013. Available: http://networkx.github.io/

[17] *Python*. Python Software Foundation, 2013. Available: http://www.python.org/

[18] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Ph.D. dissertation, Elect. Res. Lab., Univ. of California, Berkeley, CA, 1975.

[19] A. Mądry, "From Graphs to Matrices, and Back: New Techniques for Graph Algorithms," Ph.D. dissertation, EECS, MIT, Cambridge, MA, 2011.

[20] *Ansoft Q3D Extractor*. ANSYS, 2010. Available: http://www.ansys.com/Products/Simulation+Technology/Electromagnetics/Signal+Integrity/ANSYS+Q3D+Extractor/

[21] G. S. Smith, "Proximity Effect in Systems of Parallel Conductors," *Journal of Applied Physics*, vol. 43, no. 5, p. 2196, 1972.

[22] K. Miettinen, *Nonlinear Multiobjective Optimization*. Springer, 1999.

[23] C.-L. Hwang and A. S. M. Masud, *Multiple objective decision making, methods and applications: a state-of-the-art survey*. Springer-Verlag, 1979.

[24] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Struct Multidisc Optim*, vol. 26, no. 6, pp. 369–395, Apr. 2004.

[25] N. Hingora, "Power-CAD: A Tool for Analysis and Optimization of Power Electronic Module Layouts," University of Arkansas, Fayetteville, 2009.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182 –197, Apr. 2002.

[27] C. A. Coello Coello, "Evolutionary multi-objective optimization: a historical view of the field," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 28–36, 2006.

[28] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jul. 2012.

[29] N. Weste, "Virtual grid symbolic layout," in *Proceedings of the 18th Design Automation Conference*, Piscataway, NJ, USA, 1981, pp. 225–233.

[30] *PySide: Python for Qt*. 2013. Available: http://qt-project.org/wiki/PySide

[31] *Qt: Cross-Platform UI Framework*. Digia, 2012. Available: http://qt-project.org/

[32] *matplotlib: Python Plotting*. 2013. Available: http://matplotlib.org/

[33] S. Li, L. M. Tolbert, F. Wang, and F. Z. Peng, "P-cell and N-cell based IGBT module: Layout design, parasitic extraction, and experimental verification," in *Applied Power Electronics Conference and Exposition (APEC), 2011 Twenty-Sixth Annual IEEE*, pp. 372–378.

[34] P. L. Evans, A. Castellazzi, and C. M. Johnson, "Automated fast extraction of compact thermal models for power electronic modules," *IEEE Transactions on Power Electronics*, vol. 28, no. 10, pp. 4791–4802, Oct. 2013.

[35] M. Reyes-Sierra and C. A. . Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *International Journal of Computational Intelligence Research*, vol. 2, no. 3, pp. 287–308, 2006.

Appendix A: GEO Script Example

```
Function BoxSurfaces
   // Box Points
   // w: half width of the box (x-axis)
   // l: half length of the box (y-axis)
   // t: thickness of the box (z axis)
   // lc: characteristic mesh length
   p1 = newp; Point(p1) = {-w, -l,  0, lc};
   p2 = newp; Point(p2) = {w, -l,  0, lc};
   p3 = newp; Point(p3) = {w, l, 0, lc};
   p4 = newp; Point(p4) = {-w, l, 0, lc};

   p5 = newp; Point(p5) = {-w, -l, t, lc};
   p6 = newp; Point(p6) = {w, -l, t, lc};
   p7 = newp; Point(p7) = {w, l, t, lc};
   p8 = newp; Point(p8) = {-w, l, t, lc};

   l1 = newl; Line(l1) = {p1,p2};
   l2 = newl; Line(l2) = {p2,p3};
   l3 = newl; Line(l3) = {p3,p4};
   l4 = newl; Line(l4) = {p4,p1};

   l5 = newl; Line(l5) = {p5,p6};
   l6 = newl; Line(l6) = {p6,p7};
   l7 = newl; Line(l7) = {p7,p8};
   l8 = newl; Line(l8) = {p8,p5};

   l9 = newl; Line(l9) = {p1,p5};
   l10 = newl; Line(l10) = {p2,p6};
   l11 = newl; Line(l11) = {p3,p7};
   l12 = newl; Line(l12) = {p4,p8};

   // Bottom
   s1 = news; Line Loop(s1) = {l1,l2,l3,l4}; Plane Surface(s1) = {s1};
   // Top
   s2 = news; Line Loop(s2) = {l5,l6,l7,l8}; Plane Surface(s2) = {s2};
   // Sides
   s3 = news; Line Loop(s3) = {l1,l10,-l5,-l9}; Plane Surface(s3) = {s3};
   s4 = news; Line Loop(s4) = {l2,l11,-l6,-l10}; Plane Surface(s4) = {s4};
   s5 = news; Line Loop(s5) = {l3,l12,-l7,-l11}; Plane Surface(s5) = {s5};
   s6 = news; Line Loop(s6) = {l4,l9,-l8,-l12}; Plane Surface(s6) = {s6};
Return
```

```
widths[] = { 0.1,0.095,0.095,0.095,0.085,0.0048,0.0048 };
lengths[] = { 0.08,0.075,0.075,0.075,0.065,0.0024,0.0024 };
thicks[] = { 0.003,0.0001,0.00041,0.00064,0.00041,0.0001,0.00035 };
lcs[] = { 0.00266666666667,0.0025,0.0025,0.0025,0.0025,0.00024,0.00024 };

bottom_surf = -1; // 1
top_surf = -1;    // 2
ext_surfs[] = {}; // 3
int_surfs[] = {}; // 4

vol_offset = 1;
last_index = #widths[]-1;

ext_surf_cnt = 0;
int_surf_cnt = 0;
layer_surf_cnt = 0;

cur_z = 0.0;
For i In {0:last_index}
   w = widths[i]*0.5;
   l = lengths[i]*0.5;
   t = thicks[i];
   lc = lcs[i];
   Call BoxSurfaces;
   Translate {0, 0, cur_z} {Surface{s1,s2,s3,s4,s5,s6};}

   // Add side surfs to ext_surfs list
   ext_surfs[ext_surf_cnt] = s3;
   ext_surf_cnt += 1;
   ext_surfs[ext_surf_cnt] = s4;
   ext_surf_cnt += 1;
   ext_surfs[ext_surf_cnt] = s5;
   ext_surf_cnt += 1;
   ext_surfs[ext_surf_cnt] = s6;
   ext_surf_cnt += 1;

   // If not at the last index delete top surface
   If (i<last_index)
      Delete {Surface{s2};}
   EndIf

   If (i > 0)
      If (w < prev_w && l < prev_l)
         il1 = newl; Line(il1) = {prev_pts[0], p1};
         il2 = newl; Line(il2) = {prev_pts[1], p2};
```

```
        il3 = newl; Line(il3) = {prev_pts[2], p3};
        il4 = newl; Line(il4) = {prev_pts[3], p4};
        is1 = news; Line Loop(is1) = {prev_lns[0],il2,-l1,-il1};
        Plane Surface(is1) = {is1};
        is2 = news; Line Loop(is2) = {prev_lns[1],il3,-l2,-il2};
        Plane Surface(is2) = {is2};
        is3 = news; Line Loop(is3) = {prev_lns[2],il4,-l3,-il3};
        Plane Surface(is3) = {is3};
        is4 = news; Line Loop(is4) = {prev_lns[3],il1,-l4,-il4};
        Plane Surface(is4) = {is4};
        Surface Loop(i) = {prev_surfs[0],prev_surfs[2],prev_surfs[3],
                        prev_surfs[4],prev_surfs[5],is1,is2,is3,is4,s1};

        ext_surfs[ext_surf_cnt] = is1;
        ext_surf_cnt += 1;
        ext_surfs[ext_surf_cnt] = is2;
        ext_surf_cnt += 1;
        ext_surfs[ext_surf_cnt] = is3;
        ext_surf_cnt += 1;
        ext_surfs[ext_surf_cnt] = is4;
        ext_surf_cnt += 1;

        //Physical Surface(5+layer_surf_cnt) = {is1, is2, is3, is4, s1};
        //layer_surf_cnt += 1;
    EndIf

    If (w == prev_w && l == prev_l)
        Surface Loop(i) = {prev_surfs[0],prev_surfs[2],prev_surfs[3],
                        prev_surfs[4],prev_surfs[5],s1};

        //Physical Surface(5+layer_surf_cnt) = {s1};
        //layer_surf_cnt += 1;
    EndIf

    Volume(i) = {i};
    Physical Volume(i) = {i};

    int_surfs[int_surf_cnt] = s1;
    int_surf_cnt += 1;
EndIf

// Bottom Box
If (i == 0)
    bottom_surf = s1;
    //Physical Surface(5+layer_surf_cnt) = {s1};
```

```
      //layer_surf_cnt += 1;
   EndIf

   // Middle Boxes
   If (i > 0 && i < last_index)
   EndIf

   // Top Box
   If (i == last_index)
      Surface Loop(i+1) = {s1,s2,s3,s4,s5,s6};
      Volume(i+1) = {i+1};
      Physical Volume(i+1) = {i+1};
      top_surf = s2;

      //Physical Surface(5+layer_surf_cnt) = {s2};
      //layer_surf_cnt += 1;
   EndIf

   cur_z += t;
   prev_w = w; prev_l = l; prev_t = t;
   prev_pts[] = {p5,p6,p7,p8};
   prev_lns[] = {l5,l6,l7,l8};
   prev_surfs[] = {s1,s2,s3,s4,s5,s6};
EndFor

Physical Surface(1) = {bottom_surf};
Physical Surface(2) = {top_surf};
Physical Surface(3) = ext_surfs[];
Physical Surface(4) = int_surfs[];
```

Appendix B: SIF File Example

```
CHECK KEYWORDS Warn

Header
  Mesh DB "." "thermal_char"
End

Simulation
  Min Output Level = 0
  Max Output Level = 31
  Output Caller = True

  Coordinate System = "Cartesian 3D"
  Coordinate Mapping(3) = 1 2 3

  Simulation Type = "Steady State"
  Steady State Max Iterations = 1
  Output Intervals = 1

  Output File = "data.dat"
  Post File = "data.ep"
End

Constants
  Gravity(4) = 0 -1 0 9.82
  Stefan Boltzmann = 5.67e-08
End

Body 1
  Name = "baseplate"
  Equation = 1
  Material = 1
End

Body 2
  Name = "sub_solder"
  Equation = 1
  Material = 2
End

Body 3
  Name = "metal2"
```

```
  Equation = 1
  Material = 3
End

Body 4
  Name = "isolation"
  Equation = 1
  Material = 4
End

Body 5
  Name = "metal1"
  Equation = 1
  Material = 3
End

Body 6
  Name = "die_solder"
  Equation = 1
  Material = 5
End

Body 7
  Name = "device"
  Equation = 1
  Material = 6
End

Equation 1
  Name = "Equation1"
  Heat Equation = True
  Active Solvers(2) = 1 2
End

Solver 1
  Exec Solver = "Always"
  Equation = "Heat Equation"
  Variable = "Temperature"
  Variable Dofs = 1
  Linear System Solver = "Iterative"
  Linear System Iterative Method = "TFQMR"
  Linear System Max Iterations = 500
  Linear System Convergence Tolerance = 1.0e-9
  Linear System Abort Not Converged = True
  Linear System Preconditioning = "ILU0"
```

```
  Linear System Residual Output = 1
  Steady State Convergence Tolerance = 1.0e-09
  Stabilize = True
  Nonlinear System Convergence Tolerance = 1.0e-05
  Nonlinear System Max Iterations = 1
  Nonlinear System Newton After Iterations = 3
  Nonlinear System Newton After Tolerance = 1.0e-02
  Nonlinear System Relaxation Factor = 1.0
End

Solver 2
  Exec Solver = "Always"
  Equation = "Flux Compute"
  Procedure = "FluxSolver" "FluxSolver"
  Average Within Materials = Logical True
  Calculate Flux = Logical True
  Linear System Solver = "Iterative"
  Linear System Iterative Method = "cg"
  Linear System Preconditioning = ILU0
  Linear System Residual Output = 10
  Linear System Max Iterations = Integer 500
  Linear System Convergence Tolerance = 1.0e-10
End

Material 1
  Name = "Baseplate_Mat"
  Density = 9700.0
  Heat Conductivity = 190.0
End

Material 2
  Name = "SubAttach_Mat"
  Density = 11020.0
  Heat Conductivity = 35.8
End

Material 3
  Name = "Trace_Mat"
  Density = 2710.0
  Heat Conductivity = 200.0
End

Material 4
  Name = "Iso_Mat"
  Density = 3250.0
```

```
  Heat Conductivity = 160.0
End

Material 5
  Name = "DieAttach_Mat"
  Density = 11020.0
  Heat Conductivity = 35.8
End

Material 6
  Name = "Device_Mat"
  Density = 3200.0
  Heat Conductivity = 120.0
End

Boundary Condition 1
  Name = "DeviceLoad"
  Target Boundaries(1) = 2
  Heat Flux BC = Logical True
  Heat Flux = 2604166.66667
End

Boundary Condition 2
  Name = "Cooling"
  Target Boundaries(1) = 1
  Heat Flux BC = Logical True
  External Temperature = 300.0
  Heat Transfer Coefficient = 100.0
End
```

Appendix C: Vertical Design Variable Analysis Pseudocode

```
Input Parameter: trace_matrix                    # Trace specific matrix (2D) input
dv_count = prev_dv_count                         # Init. design variable count to
previous count
row_list = { }                                   # Create empty row list
N = x_length(trace_matrix)                       # Number of columns in trace matrix
M = y_length(trace_matrix)                       # Number of rows in trace matrix

for x = 0 to M-1:                                # Iterate over all rows in matrix
    # Build list of unique horizontal traces in current row
    horz_traces = { }                            # Holds unique horizontal trace objects
    for y = 0 to M-1:                            # Iterate over the length of the row
        for each trace in trace_matrix[x][y]:        # Get all traces in current position
            if  (trace is horizontal) and (trace not in horz_traces):
                # Trace is unique and horizontal, append to horz_traces list
                horz_traces.append(trace)

    dv_indices = { }                             # Indices of design variable list
    if length(horz_traces) == 0:                 # Empty row (Phantom)
        # Row represents empty space,
        # Assign a design variable to width of row
        dv_indices.append(dv_count)
        dv_count = dv_count + 1
    else if length(horz_traces) >= 1:            # Single or multiple trace row
        # Give each horizontal trace object in the row a design variable
        for each trace in horz_traces:
            dv_indices.append(dv_count)
            dv_count = dv_count + 1
    row = {horz_traces, dv_indices}              # Create row object
    row_list.append(row)                         # Append row object to list
```

Appendix D: Pseudocode for all Data Structures at all Levels (Except Devices)

Pseudocode for baseplate library level data structure:

```
class Baseplate:
    properties          # Material properties object for baseplate: (Reference)
```

Pseudocode for baseplate project level data structure:

```
class BaseplateInstance:
    dimensions          # Dimensions [width, length, thickness]: mm (List of floats)
    eff_conv_coeff      # Effective Convection Coefficient: W·m^-2 (float)
    baseplate_tech      # Baseplate library level object: (Reference)
```

Pseudocode for substrate library level data structure:

```
class Substrate:
    name                # Part name: (String)
    isolation_properties # Material properties object for baseplate: (Reference)
    metal_properties    # Material properties object for baseplate: (Reference)
    metal_thickness     # Metal thickness: mm (float)
    isolation_thickness # Dielectric isolation thickness: mm (float)
```

Pseudocode for substrate project level data structure:

```
class SubstrateInstance:
    dimensions          # Dimensions [width, length]: mm (List of floats)
    ledge_width         # Width of metal removed around border of substrate: mm (float)
    substrate_tech      # Substrate library level object: (Reference)
```

Pseudocode for substrate attach library level data structure:

```
class SubstrateAttach:
    properties          # Material properties object: (Reference)
```

Pseudocode for substrate attach project level data structure:

```
class SubstrateAttachInstance:
    thickness       # Thickness of attach layer: mm (float)
    attach_tech     # SubstrateAttach library level object: (Reference)
```

Pseudocode for die attach library level data structure:

```
class DieAttach:
    properties      # Material properties object: (Reference)
```

Pseudocode for bondwire library level data structure:

```
class BondWire:
    name            # Part name: (String)
    wire_type       # Type of bond wire; POWER =1, SIGNAL=2: (int)
    shape           # Shape of bond wire; ROUND=1, RIBBON=2: (int)
    dimensions      # If shape is 'round': diameter mm (float)
                    # If shape is 'ribbon': [width, thickness] mm (List of floats)
    a               # Height of bond wire from the die: mm (float)
    b               # Fraction of bond wire total length: unitless (float)
    properties      # Material properties object: (Reference)
```

Pseudocode for bondwire solution level data structure:

```
class BondwireSolution:
    positions       # Start and stop positions of bond wire [[x0, y0], [x1, y1]] (floats)
    beg_height      # Height from top of trace to start point of bondwire: mm (float)
    height          # Max height of bondwire w.r.t. top of die: mm (float)
    bondwire_tech   # Bondwire library level object: (Reference)
    eff_diameter    # Effective diameter of bondwire: mm (float)
```

Pseudocode for lead library level data structure:

```
class Lead:
        name            # Part name: (String)
        lead_type       # Type of lead; POWER=1, SIGNAL=2: (int)
        shape           # Shape of lead; BUSBAR=1, ROUND=2: (int)
        dimensions      # If shape is BUSBAR: [w, l, t, h]: mm (floats)
                        # If shape is ROUND:  [diameter, height]: mm (floats)
        properties      # Material properties object: (Reference)
```

Pseudocode for lead solution level data structure:

```
class LeadSolution:
        position        # Center position of lead [x0, y0]: mm (floats)
        orientation     # Lead orientation; Top=1, Bottom=2, Right=3, Left=4: (int)
        lead_tech       # Lead library level object
```

Appendix E: All Technology Library Editor Pages (Except Device Page)

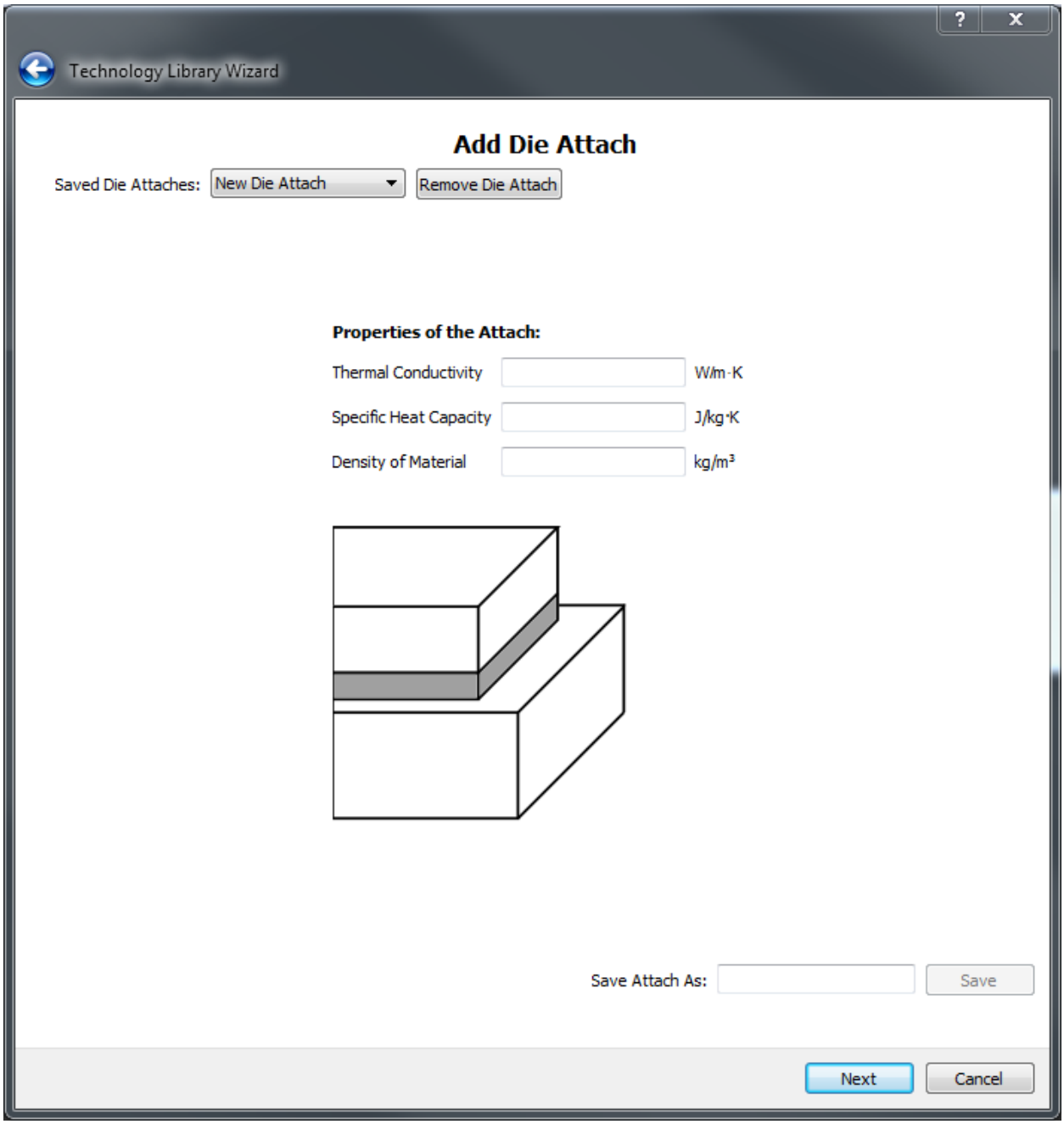


Fig. E.1: TLE Die Attach page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

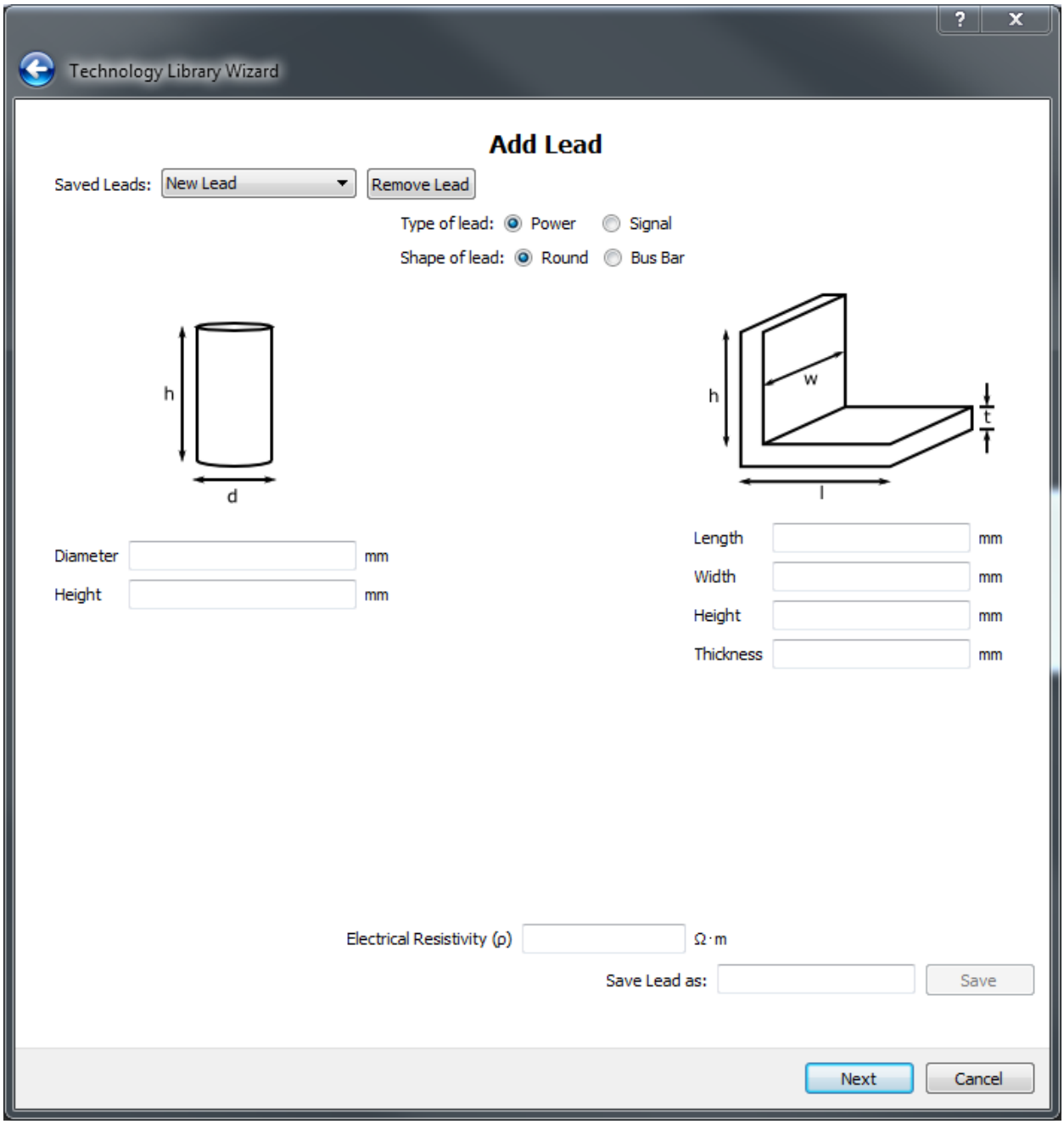


Fig. E.2: TLE Lead page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

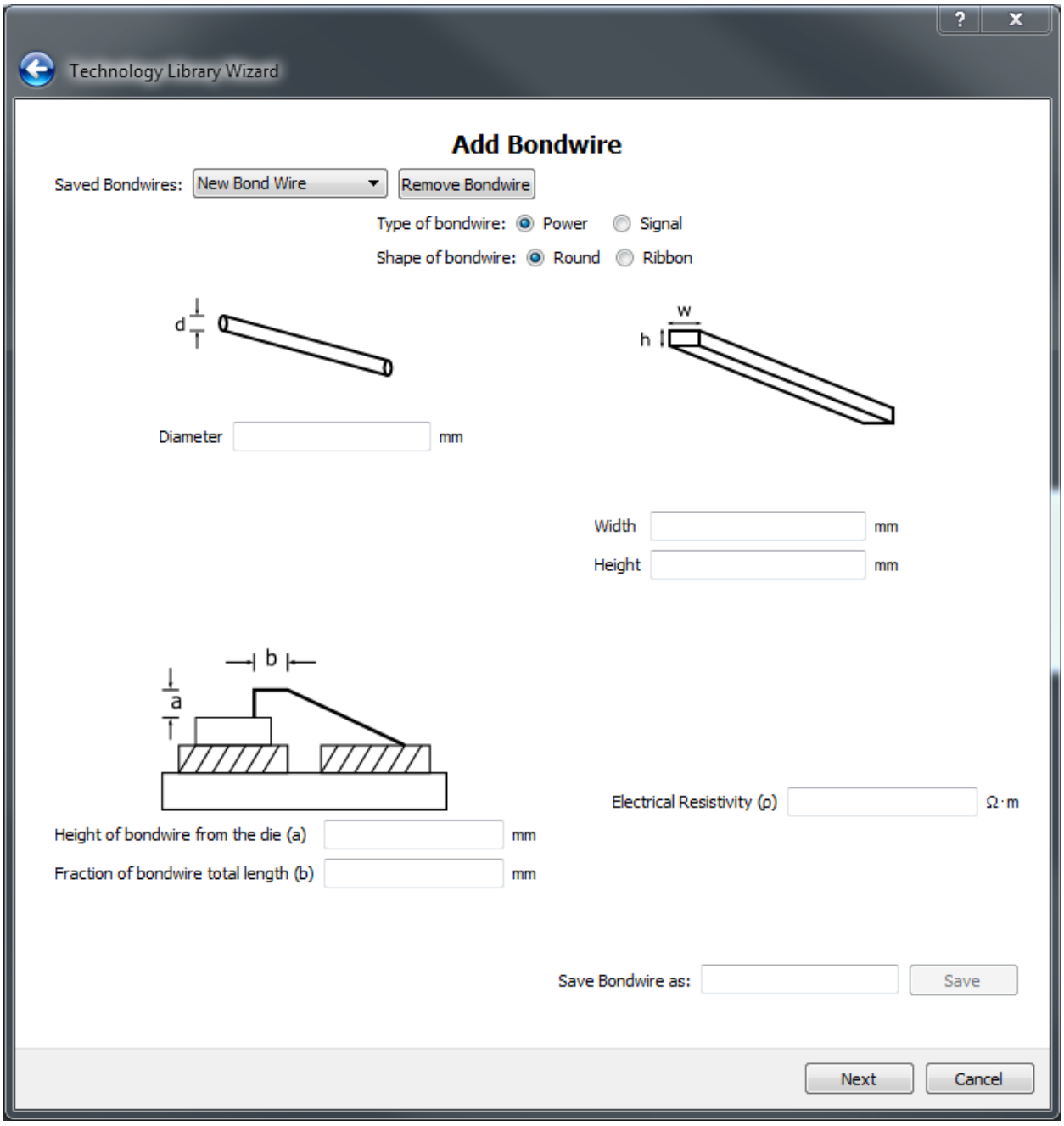


Fig. E.3: TLE Bondwire page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

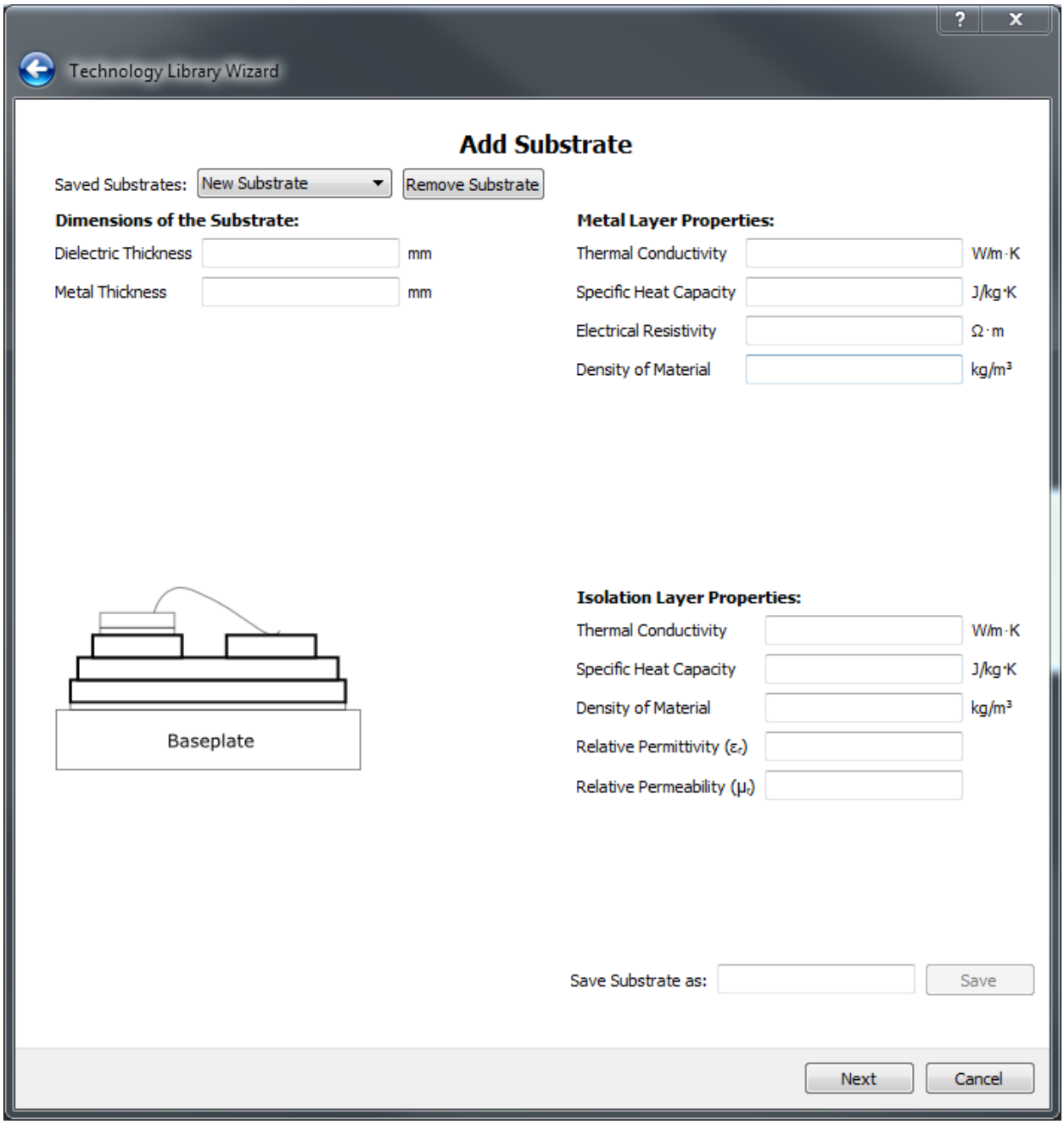


Fig. E.4: TLE Substrate page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

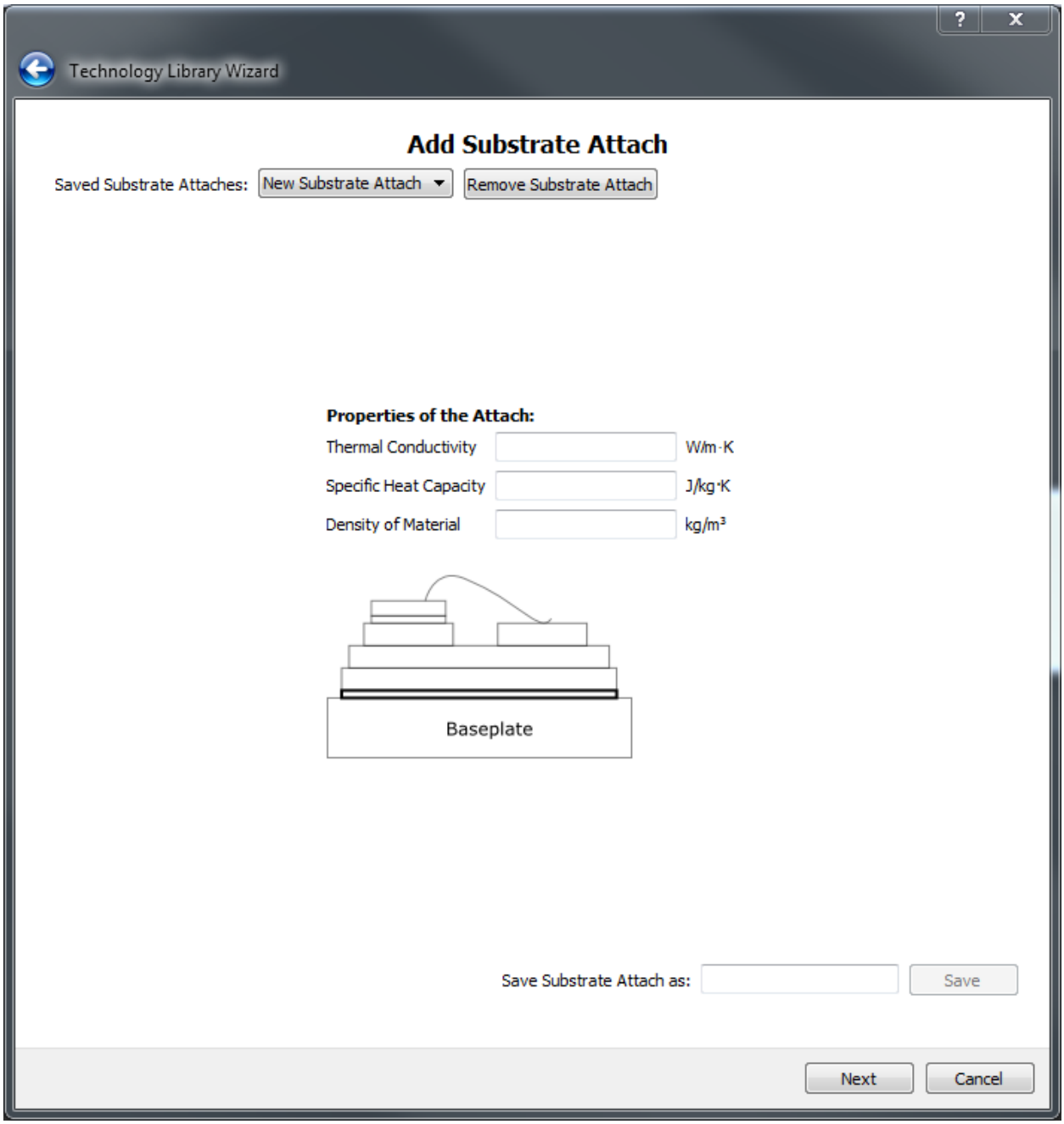


Fig. E.5: TLE Substrate Attach page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

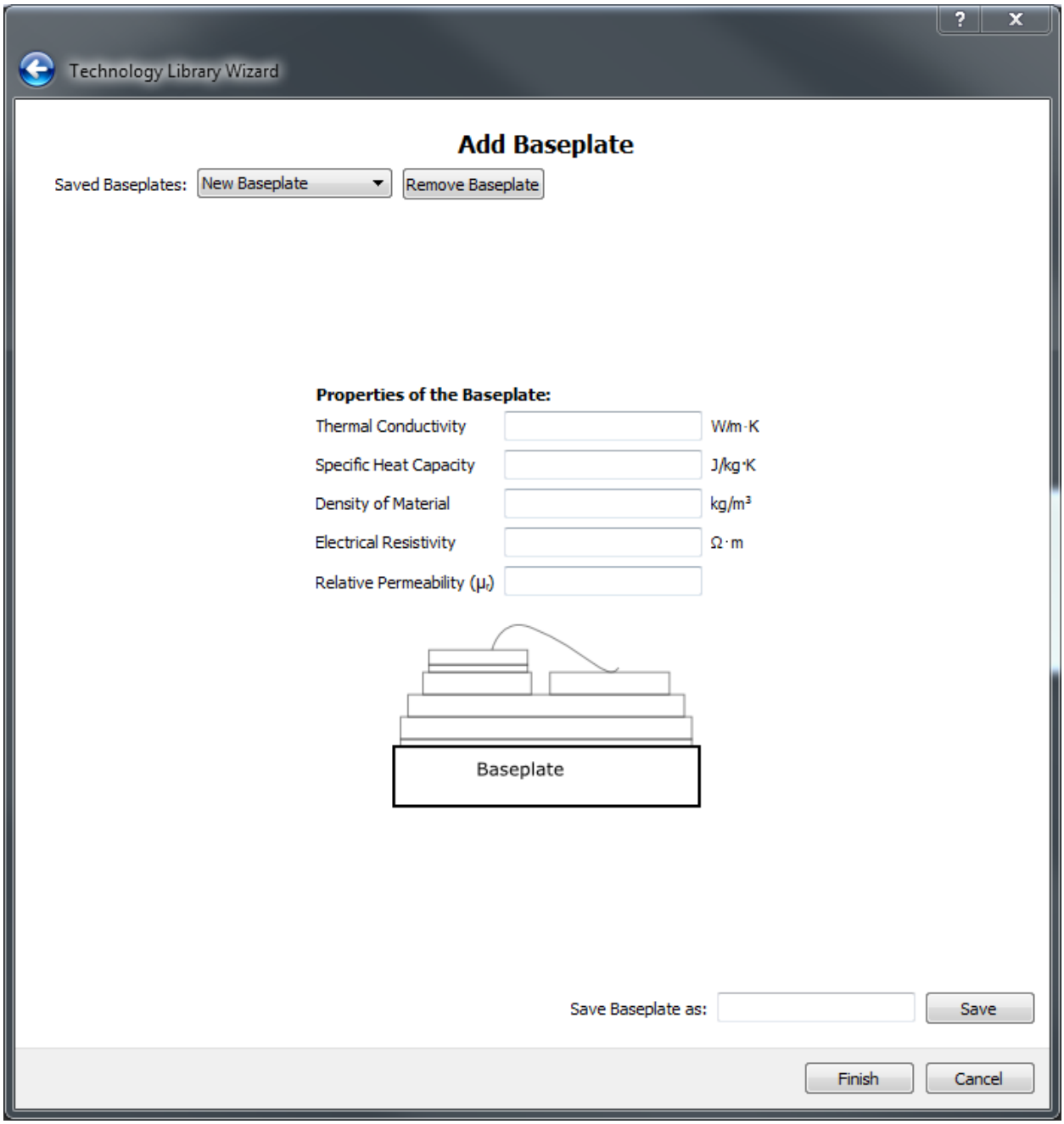


Fig. E.6: TLE Baseplate page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

Appendix F: Project Builder Pages (Omitted from Writing)

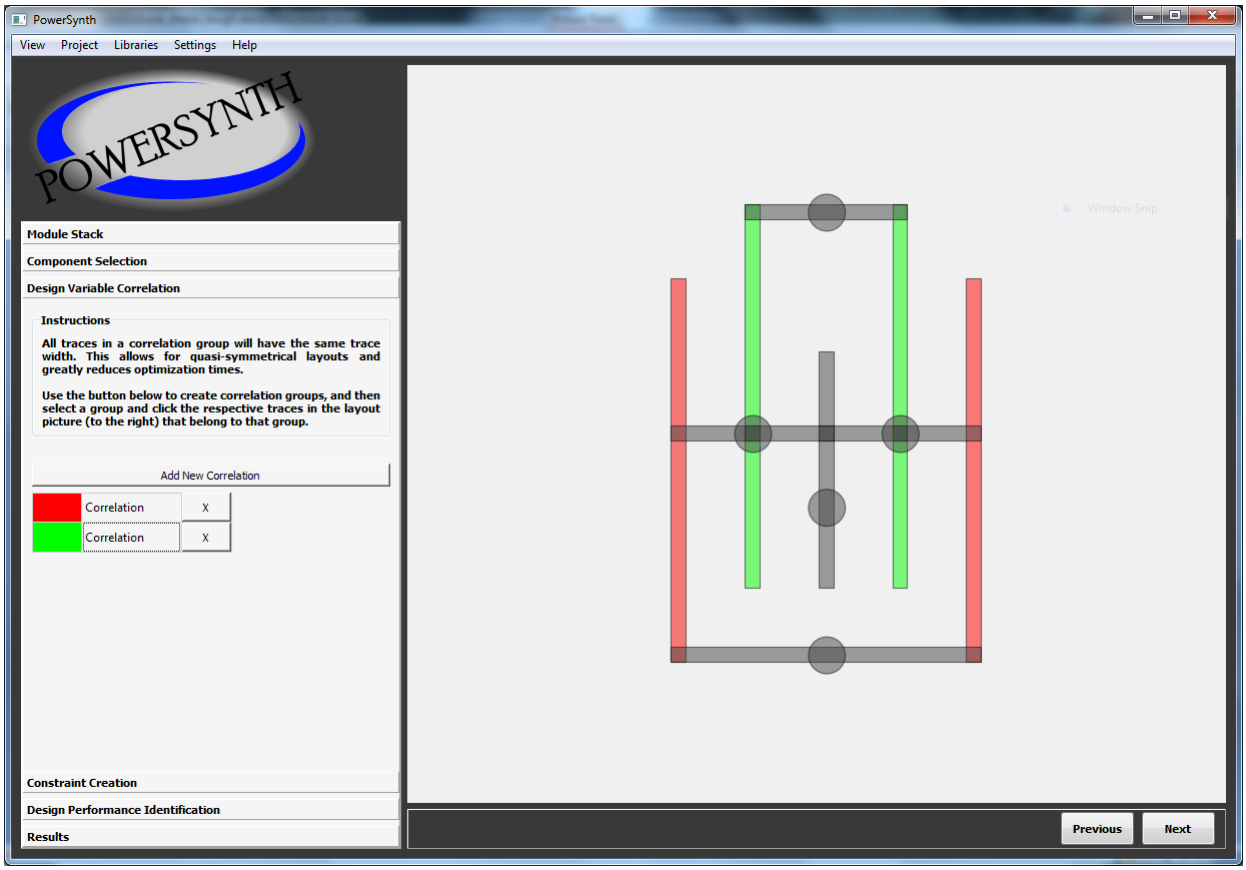


Fig. F.1: Design Variable Correlation page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

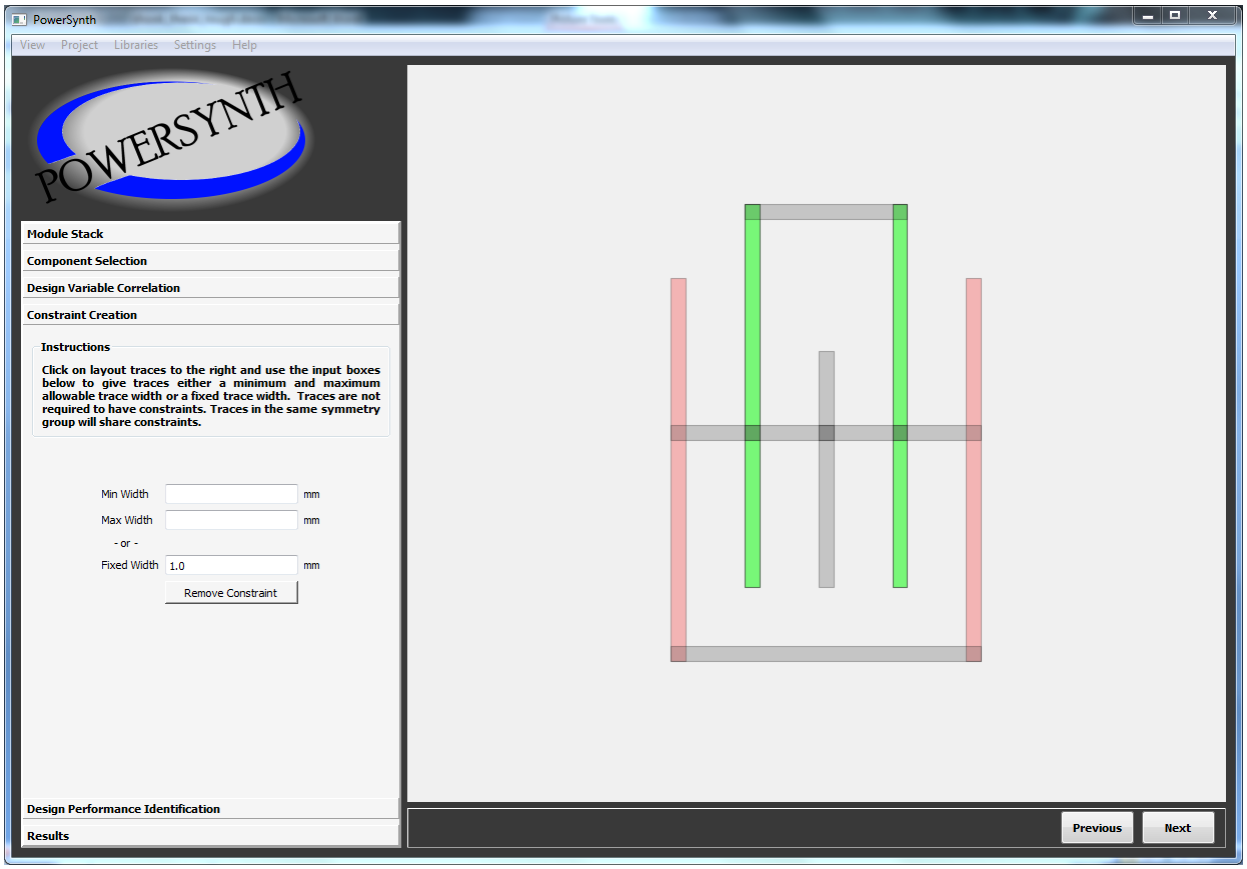


Fig. F.2: Constraint Creation page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

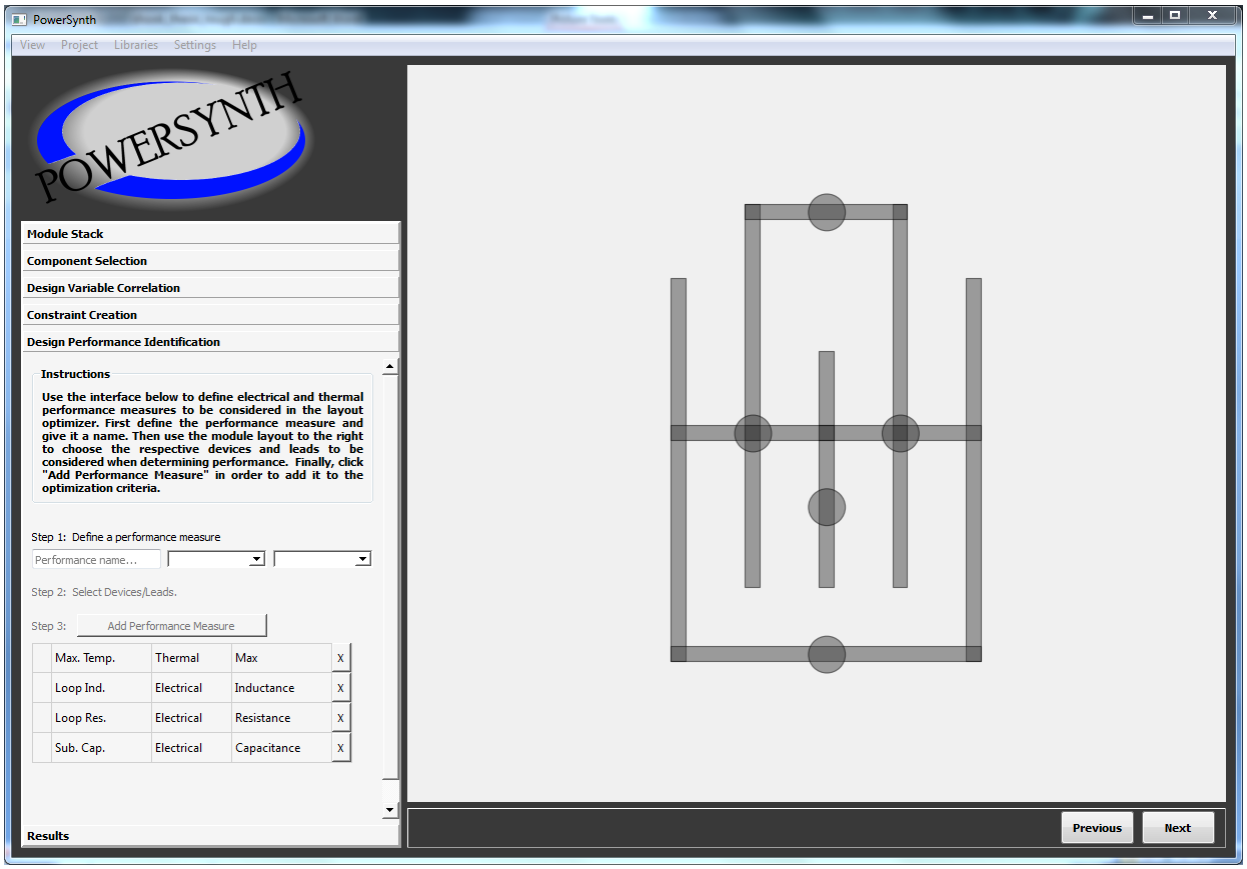


Fig. F.3: Design Performance Identification page (the above software screenshot is intended only to give the reader a sample view of the software discussed).

Appendix G: Moore-Penrose Pseudoinverse of Matrix Example

This section contains an example of the Moore-Penrose pseudoinverse. The Laplacian matrix of a simple circuit is found and its pseudoinverse is found. A unitary flow of current is connected to the circuit and the resulting voltage between two nodes is solved for using the matrix pseudoinverse. There are several methods for finding the Moore-Penrose pseudoinverse such as singular value decomposition (SVD). The internal workings of these methods are not shown here, but may be readily found.

Fig. G.1 shows the simple circuit used for the example. The goal here is to measure the voltage from node 2 to node 0. Notice, that no ground node is required to be chosen. The pseudoinverse technique allows for no reference node to be established. The Laplacian of the matrix **Y** is generated from the circuit by an element stamping technique.

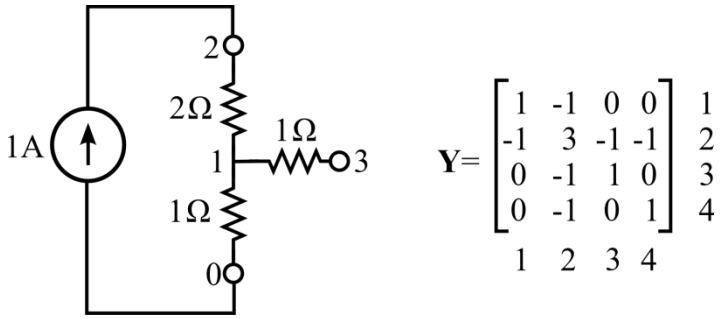


Fig. G.1: Simple circuit and its Laplacian matrix.

Eq. (G.1) represents the relationship between a vector of node voltages **v** and a current source vector **j**.

$$\mathbf{Yv=j} \tag{G.1}$$

The matrix inverse of **Y** needs to be found in order to solve for **v** given **j** (Eq. (G.2)), but **Y** is not invertible or ill-conditioned. SPICE removes the row and column associated with the ground node to create an admittance matrix which is invertible. In this case, it is desired not to remove the ground node row and column. The pseudoinverse of **Y** ($\mathbf{Y}^{\dagger}$) is found instead. Fig. G.2 shows the numerical results of this process.

$$\mathbf{v}=\mathbf{Y}^{\dagger}\mathbf{j} \tag{G.2}$$

$$\mathbf{Y}^{\dagger}=\begin{bmatrix} 0.75 & 0 & -0.5 & -0.25 \\ 0 & 0.25 & -0.25 & 0 \\ -0.5 & -0.25 & 1.25 & -0.5 \\ -0.25 & 0 & -0.5 & 0.75 \end{bmatrix}\begin{matrix}0\\1\\2\\3\end{matrix}$$
$$\begin{matrix} 0 & 1 & 2 & 3 \end{matrix}$$

Fig. G.2: Moore-Penrose pseudoinverse results.

The results of solving Eq. (G.2) for **v** is shown in Fig. G.3. Notice that v(2)-v(0) = 3 V. The voltage between any node in the circuit may be found in this manner without a specific ground reference.

$$\mathbf{j}=\begin{bmatrix}-1\\0\\1\\0\end{bmatrix}\begin{matrix}0\\1\\2\\3\end{matrix} \longrightarrow \mathbf{v}=\begin{bmatrix}-1.25\\-0.25\\1.75\\-0.25\end{bmatrix}\begin{matrix}0\\1\\2\\3\end{matrix}$$

Fig. G.3: Current vector and voltage output vector.