Hierarchical Layout Synthesis and Optimization Framework for High-Density Power Module Design Automation

Imam Al Razi[a], Quang Le[b], H. Alan Mantooth[b], Yarui Peng[a]
[a] Computer Science and Computer Engineering Department, [b] Electrical Engineering Department
University of Arkansas, Fayetteville, AR
ialrazi@uark.edu, yrpeng@uark.edu

***Abstract*—Multi-chip power module (MCPM) layout design automation has become an emerging research field in the power electronics society. MCPM physical design is currently a trial-and-error procedure that heavily relies on the designers' experience to produce a reliable solution. To push the boundary of energy efficiency and power density, novel packaging technologies are emerging with increasing design complexity. As this manual design process becomes the bottleneck in design productivity, the power electronics industry is calling for more intelligence in design CAD tools, especially for advanced packaging solutions with stacked substrates. This paper presents a physical design, synthesis, and optimization framework for 2D, 2.5D, and 3D power modules. Generic, scalable, and efficient physical design algorithms are implemented with optimization metaheuristics to solve the hierarchical layout synthesis problem. Corner stitching data structure and hierarchical constraint graph evaluation have been customized to better align with power electronics design considerations. A complete layout synthesis process is demonstrated for both 2D and 3D power module examples. Further, electro-thermal design optimization is carried out on a sample 3D MCPM layout using both exhaustive and evolutionary search methods. Our algorithm can generate 937 3D layouts in 56 s, resulting in 10 layouts on the Pareto-front. In addition, our optimized 3D layouts can achieve 1.3 nH loop inductance with 38 °C temperature rise and 836 mm^2 footprint area, compared to 2D layouts with 8.5 nH, 99 °C, and 2000 mm^2.**

***Keywords*—PowerSynth, 2D/2.5D/3D Power Module, Layout Optimization, Physical Design Automation**

I. INTRODUCTION

With the advent of wide bandgap (WBG) power semiconductor devices (i.e., SiC, GaN), multi-chip power module performance has been improved significantly [1]. Recent advances in the power electronics industry have enabled power conversion design with enhanced efficiency, compact physical structure, higher reliability [2]. Therefore, the traditional, iterative design approach is unable to satisfy the ever-growing demands for optimized power modules with high power density. To reduce engineering time and cost, the industry is looking for electronic design automation (EDA) tools. In the meantime, researchers are shifting their concentration to develop advanced algorithms for emerging packaging technologies [3].

Researchers from both analog/mixed-signal and power electronics societies have been adapting the design automation methodologies from VLSI as the EDA tools for digital IC design are highly matured compared to the others [4]. The similarity between VLSI and analog computer-aided design (CAD) flow has led a large group of researchers towards developing different CAD tools for analog/mixed-signal layout design automation to address critical challenges like handling device parameterization, constraint generation, maintaining symmetrical placement and routing, etc. [4–7]. Unlike analog and digital CAD flow, Power-CAD [8] requires simultaneous analysis of thermal, electrical, and mechanical parameters to design an optimal layout. Therefore, layout compaction is not always the best approach for power modules to ensure reliable operation. Recent works on power module layout optimization have adapted some of the VLSI concepts to introduce automated design flow. The sequence pair representation technique has been used in [9] for representing and optimizing placement of the components in a power module layout. A 1D binary string is used to optimize routing paths. Though this approach has been able to generate solutions automatically, a simplified representation of components leads to inefficiency with handling complex geometry. In [10], a matrix-based methodology is used in the MCPM layout generation tool called PowerSynth to generate layout solutions during optimization. This methodology leads to iterative DRC-checking, limited solution space, and long run time. To overcome the limitations with the matrix-based methodology, a more generic, efficient, and scalable approach based on corner stitching data structure with a hierarchical constraint graph methodology has been implemented by replacing the matrix-based layout engine [11]. This approach has successfully demonstrated a 2.5D power module layout optimization. To show the concept of 3D layout design, the connection handling algorithms are demonstrated in [12]. Though the methodology has been proven to be beneficial for optimizing 3D MCPM layouts, it cannot optimize wire bondless 3D geometry. This type of geometry handling requires an arbitrary depth of fixed constraint handling and honoring constraints that are shared across different layers. Also, rather than using any optimization algorithm, exhaustive searching through randomization has been performed to

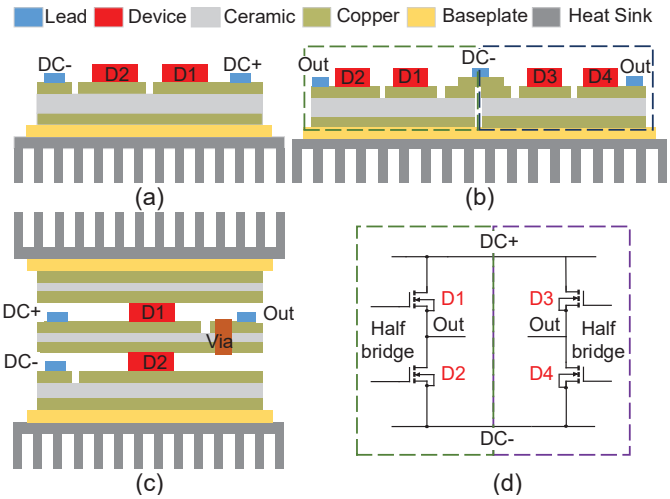

Fig. 1: Cross sections of MCPM structures: (a) 2D half-bridge, (b) 2.5D full-bridge, (c) 3D half-bridge, and (d) full-bridge power module circuit.

This material is based on work supported by The National Science Foundation under Grant No. EEC-1449548 and Army Research Lab Contract No. W911NF1820087. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Science Foundation and Army Research Lab.

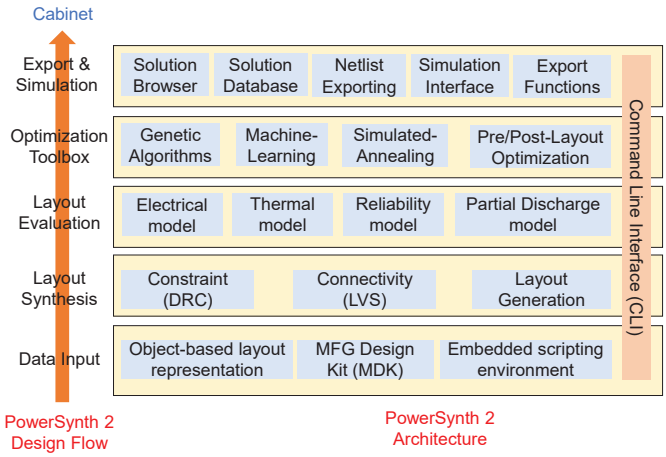

Fig. 2: Power electronics design automation framework architecture

find the tradeoff between electrical and thermal performances.

To address these limitations, an updated hierarchical, generic, and scalable framework for 2D, 2.5D, 3D MCPM layout optimization has been presented in this paper. The definitions of three different layouts within this framework are as follows. 2D layout refers to a single device layer, as shown in Fig. 1(a). A full-bridge MCPM circuit consisting of two half-bridge circuits is shown in Fig. 1(d). Here, Fig. 1(b) represents a 2.5D layout, which is defined by multiple routing but a single device layer on the same substrate. Finally, Fig. 1(c) shows a 3D half-bridge module that consists of multiple devices and multiple routing layers stacked vertically. Due to the 3D stacking, the electrical performance has been improved, but thermal management has become a challenge. Double-sided cooling is one solution for face-to-face stacking of the devices. However, for face-to-back stacking (shown in Fig. 1(c)), it requires at least four layers to form a half-bridge module with double-sided cooling, which increases the fabrication cost and complexity. To reduce fabrication complexity and cost, back-to-back stacking can be performed to create a half-bridge module with embedded heat sink/micro-cooler water channels between two DBCs [13]. Since different layout architectures are possible in 3D configurations, the design tool must handle all these architectures. To the best of the authors' knowledge, no existing tool can optimize all these types of high-density MCPM layouts.

In this paper, our key contributions are: (1) A physical design, synthesis, and optimization framework for 2D, 2.5D, and 3D power modules; (2) Generic, scalable, and efficient physical design algorithms for layout generation; (3) Non-dominated sorting genetic algorithm (NSGAII) [14] implementation to solve the hierarchical layout synthesis problem. The rest of the paper is organized as follows. Section II briefly introduces the tool architecture. Section III represents layout solution generation methodology along with the optimization framework. Section IV demonstrates the tool capabilities with a sample 3D MCPM layout optimization case study. Finally, Section V concludes the paper and discusses the upcoming work.

II. Layout Synthesis and Optimization Framework

Over the past decade, MCPM layout synthesis and optimization tools have been evolved by adding new underline methodologies, features, application programming interfaces (APIs), performance evaluation models, etc. The most updated architecture of PowerSynth [10] is shown in Fig. 2. Among all different parts of the proposed architecture, most features in each step of the design flow are implemented with a few ongoing. Different versions of

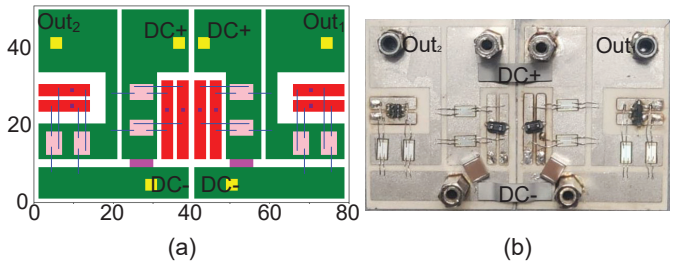

Fig. 3: (a) Balanced 2.5D solution layout from the tool, (b) fabricated layout for measurement.

PowerSynth release packages can be found in [15]. A brief overview of the current status of the architecture implementation is described here.

A. Data Input

The current version of the tool has a built-in manufacturer design kit (MDK) defining components, material properties, and dimensions for power devices, substrates, connectors, heat spreaders, wire bonds, and leads. It requires a layer stack file describing the 3D structure of the module to perform thermal performance evaluation. Typically a power module stacks a baseplate, substrate (copper/ceramic/copper), substrate-attach, die-attach, and devices, etc. into a compact footprint. This tool takes a layout geometry description script and circuit netlist as inputs for geometry and netlist information. A set of minimum constraints is also required to generate manufacturable layout solutions.

B. Layout Synthesis

The input geometry structure is stored as a hierarchical corner-stitched tree [11]. For this tree, a set of horizontal and vertical constraint graphs is generated from each node (corner-stitched plane) by using the minimum constraint values, where edge weights represent the constraint values and corner-stitched tile coordinates represent the vertices. To generate new layout solutions, these edge weights are manipulated. This constraint-aware methodology ensures the DRC-clean solution generation. Currently, for connectivity checking (LVS), the tool is dependent on user input. As long as the initial input layout is LVS-clean, all the solutions will follow that as the constraint graphs are mapped from the initial layout.

C. Layout Evaluation

Since power module performance depends on electrical, thermal, and mechanical aspects simultaneously, a legitimate tradeoff among these aspects is obvious to obtain an optimum solution. Therefore, off-the-shelf physics-based or finite element methods are not always suitable for iterative optimization as the runtime is quite large. For faster convergence, reduced-order, hardware-validated electrical, and thermal models are developed aiming at high speed with acceptable accuracy [10]. The 3D module requires more careful modeling to handle mutual coupling among different components. Research is ongoing to extend electrical models to 3D layouts. However, multi-level APIs have been developed inside the tool to leverage the existing electrical, thermal, mechanical models from other research groups or companies. For example, in this work, FastHenry [16] from MIT is used for loop inductance extraction and ParaPower [17] from Army Research Lab for static thermal evaluation.

D. Multi-Objective Optimization

High electrical parasitics can result in higher switching loss, voltage overshoots, and potentially signal integrity issues at high

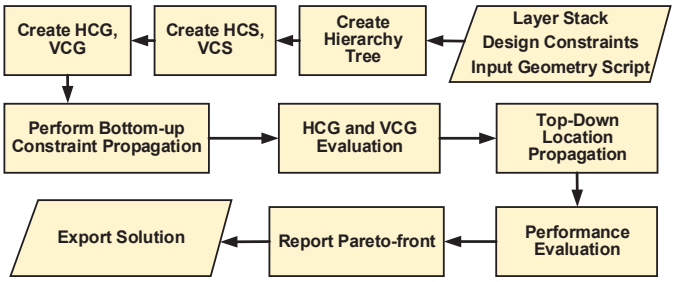

Fig. 4: High-level workflow using the built-in layout engine

switching frequencies. Likewise, bad thermal performance can lead to reliability issues of the whole system. Therefore, electrical parasitics and thermal performance are key factors of power module performance evaluation, and a cost function is defined considering both electrical and thermal parameters. Several categories of optimization algorithms are considered, including evolutionary algorithms, stochastic algorithms, and gradient-based algorithms. A genetic algorithm is selected in this work, in addition to the built-in randomization algorithm. Other types of optimization algorithms such as Machine learning and Neural Network Models are actively under research. The output of the optimizer is a Pareto-front solution set. The user can choose solutions and perform post-layout optimization and further customization, such as filleting sharp corners to reduce partial discharge threats.

E. Design Export and Simulation

This tool has a solution browser to browse each generated solution and export netlist and 3D structures to commercial tools like ANSYS-Q3D, SolidWorks. The extracted netlist can be back-annotated and compared with the input. APIs are available to perform the export function automatically.

F. Experimental Validation

PowerSynth optimization result has been validated through physical measurement for 2D/2.5D layouts. Electro-thermal optimization is performed on a 2.5D layout. The optimized solution from the solution space, and the corresponding fabricated layout are shown in Fig. 3(a), (b), respectively. Power loop inductance, and maximum junction temperature for this module are reported as 8.54 nH, and 398.28 K, respectively by PowerSynth. These results are found within 10% accuracy compared to the measurements [11].

III. Methodology

To generate layout solutions, the tool takes the initial layout, layer stack, and design constraints as input. The input information is processed through the hierarchical corner stitching data structure and constraint graph evaluation algorithms. An optimization algorithm is used to rank the solutions based on the electrical and thermal performance so that the Pareto-optimal solution set can be reported. A high-level workflow of the tool is shown in Fig. 4.

A. Input Layout

The layout engine of the tool requires a hierarchical input geometry script describing the input layout from the user. Combining with layer stack information (e.g., layer id, dimensions, type, etc.), the parser makes the input geometry compatible with the hierarchical corner stitching data structure. A sample 3D layout structure and the corresponding input script details are shown in Fig. 5, and Fig. 6, respectively. The wire bondless 3D MCPM structure (shown in Fig. 5(a)) consists of three direct bonded copper (DBC) substrates

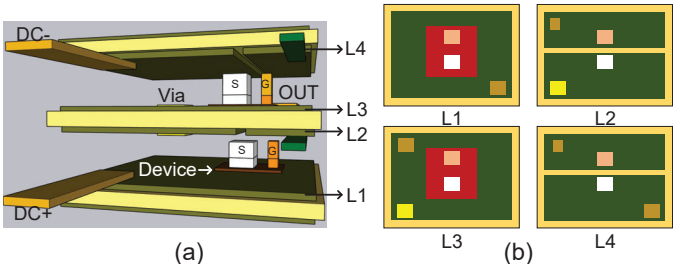

Fig. 5: (a) 3D structure of a wire bondless half-bridge module, (b) 2D layouts of each layer

(one is a through via connected), where there are four routing layers (L1-L4) and two device layers (single device per switching position). Each layer's 2D geometry is shown in Fig. 5(b). In L1, the high-side device's drain, DC+ terminal are connected. L2 contains the gate and source connections of the high-side device, the terminal for gate signal, and a landing pad of the via. L3 has the low-side device's drain, the OUT terminal, and another via landing pad. In L4, the DC- terminal, terminal for low-side gate signal, source, and gate signal pads are present. Fig. 6(a) shows the tree structure of the complete layout. The abstract and physical (the dotted rectangle) parts are separated in the illustration. The physical part is a one-to-one mapping of the 2D layouts shown in Fig. 5(b). Here, T, D, L, V represents trace, device, lead, via, respectively. The physical part is used to create the layout geometry script shown in Fig. 6(b). The abstract part saves the hierarchical information for constraint graphs, which is described in the following section. The sample geometry description script is created from the hierarchical tree representation. The first line denotes the layer name (used to map in the layer stack information) and the routing direction. The rest of the lines until the next layer name and direction describes the geometry information about each component of the same layer. The components insertion is performed hierarchically in a group-wise fashion. All connected components of the same hierarchy are in the same group. Description of a new group starts with a '+' character while '-' character denotes the continuation of the group. To declare a trace (routing component), six fields are required: name, type (power/signal), bottom-left corner's x, y coordinate, width, length. Thickness is defined in the layer stack information and is used for electrical and thermal performance models. Each 'tab' is inserted to denote the depth of the hierarchy in the tree. Each connector (lead, via) or active component (device) is declared by 4 or 5 fields. These are name, type, bottom-left corner's x, y coordinate, and orientation. In this example, the default orientation is R0. However, other orientations are defined by R90, R180, R270. Dimensions of these components are taken from the MDK as these have always fixed dimensions. To start a layout optimization using the framework, the user needs to draft the script based on the hierarchical manner, as shown in the example.

B. Input Processing

The parsed input geometry is used to create a hierarchical corner-stitched tree structure. Each node in the tree is a corner-stitched plane. For each plane, constraint graphs are created using the minimum constraint values provided by the user. A brief description of the data structure and constraint graph creation is described below.

1) Hierarchical Corner Stitching Data Structure: The basic corner stitching data structure [18] is a planar one and does not allow tile overlapping. However, for power module layout representation, overlapping of tiles needs to be allowed. Therefore, a hierarchical tree structure is maintained to store the geometry information. In

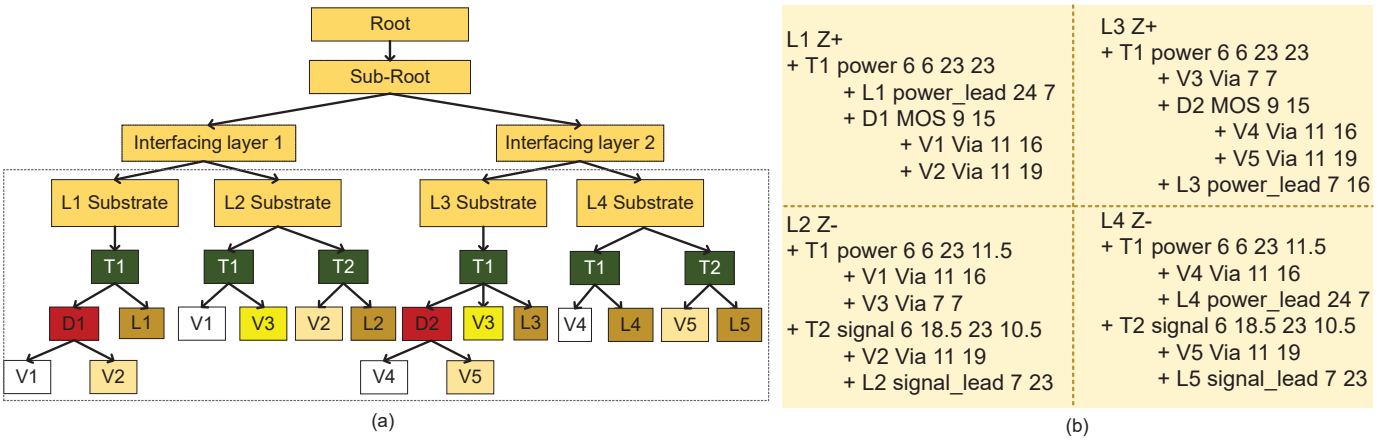

Fig. 6: (a) Tree structure of the 3D layout shown in Fig. 5 (color mapped), (b) layout geometry description script for the selected region in (a).

this tree structure, each layer of the 3D structure is started with the empty substrate as the parent node. In this node, all tiles form various groups that can be parent nodes for the next level. Since corner stitching data structure performs splitting, merging operations while inserting a tile, the non-split version of a component can be found in its parent node. For example, for the tree structure shown in Fig. 6(a), T1 is the child of the L1 substrate and at the same time parent for the device (D1) and lead (L1). The non-split version of T1 is stored in the substrate node and the device, lead is inserted on T1 in the second hierarchical level, which splits T1 but saves the non-split version of D1 and L1. Therefore, in each node of the tree, two types of tiles can be found: background and foreground. Background tiles are from parent components, while foreground tiles correspond to the newly inserted child components. Differentiation of foreground and background enables the tool to find design constraints properly. Two types of trees are maintained for each structure: Horizontal Corner-Stitched (HCS) and Vertical Corner-Stitched (VCS). All tiles are horizontally maximized in HCS, while vertically maximized in VCS. This elegant data structure is customized for power module layout representation for two reasons: (a) The tile insertion operations have linear time complexity, and (b) design constraints can be easily extracted and mapped into the constraint graphs.

2) Constraint Graph Creation: For each node in the corner-stitched tree, two constraint graphs (CGs) are created: Horizontal Constraint Graph (HCG) and Vertical Constraint Graph (VCG), which maintain relative locations among components. The constraints are found from the corner-stitched planes (HCS and VCS). Each edge in the graph corresponds to a tile in the corner-stitched plane. Thus the whole layout is mapped into HCG and VCG. However, there are some extra nodes in the constraint graph tree structure to ensure DRC-clean solution generation, which is called abstract nodes, as they do not have corresponding corner stitched planes. From Fig. 6(a), the top four nodes (outside the dotted outline) are abstract nodes. These nodes have only HCG and VCG. The interfacing layer nodes are created to handle the constraints which are shared among the layers connected with the same vias (i.e., the via-type source and gate connections of the device). For example, in the sample layout shown in Fig. 5(a), L1 and L2 layers are sharing the same device through the gate and source connections. Similarly, L3 and L4 layers have another shared device with source and gate connections. Therefore, two interfacing layers are created for two pairs of device layers. The parent node of these interfacing layers has the structure outline and via locations. This node is referred to as the sub-root node as it is the immediate child of the root node. The root node has only the structure outline coordinates. Since the abstract layers are only available in constraint graphs, these layers' HCG and VCG vertices and edges are derived from the physical layers' HCG and VCG. The current implementation restricts all layers that need to have the exact size and be perfectly aligned. However, the methodology can be easily extended to consider any offset among different layers.

Since two types of graph data structures are used in this framework to distinguish between the tree structure and the constraint graph structure, two terminologies are used: node and vertex. Here, a node refers to the hierarchical tree while a vertex the constraint graph. In each constraint graph, there may be two types of edges: (a) self edges (default type), which are generated from the corner-stitched tile, (b) propagated edges, which are propagated from the child node's constraint graph. These edges can be of two types: (a) flexible edges, which have both the source and the sink as independent vertices, (b) rigid edges, which have a dependent vertex as the sink. Four types of vertices exist in each graph: (a) source vertex with only outgoing edges, (b) sink vertex with only incoming edges, (c) independent vertex that has a fixed minimum location but no maximum location, and (d) dependent vertex that has a fixed distance to an independent vertex. All these concepts are illustrated in the sample VCGs in Fig. 8, implementing the layout hierarchy in Fig. 7. In this hierarchy, the device (L1: Node 3) is a leaf node in the L1 sub-tree, and a trace (T1) with the via-type source connection (L2: Node 3) and another trace (T2) with the via-type gate connection (L2: Node 2) makes two leaf nodes in the L2 sub-tree. For a simpler illustration, the leads and through via have not been considered. Since Node 2 and Node 3 of the L2 sub-tree have the same configuration, they have similar CGs with different vertex coordinates. The bottom graph shows the initial VCG generated by mapping the constraints from the corresponding corner-stitched planes. Here, E3 (device to via enclosure), L_V (length of the via), and L_D (length of the device) are considered as fixed constraints provided by the user and are potential rigid edges. Other constraints S3 (spacing between vias), E4 (trace to via enclosure) are non-fixed constraints. To trim the graph, the constraint validation and graph reduction algorithm (shown in Algorithm 1) is applied, and the resultant graph (Final VCG) of L1: Node 3 and L2: Node (2,3) is shown in Fig. 8(b), and (c), respectively. Algorithm 1 first identifies all vertices as candidate dependent based on the user constraints. A vertex is selected as a dependent candidate, if it is a sink of a rigid edge. Now, the stack is used to process each candidate by validating the user constraints. If any fixed constraint is valid, the dependent vertex associated with it is marked as a removable candidate as the

Algorithm 1: Constraint validation and graph reduction

Input : node CG, user constraints
Output: trimmed CG

```
select candidate vertices and edges for removal
push all candidate vertices into a stack
while stack is not empty do
    current vertex (vi) = stack.pop()
    determine the reference vertex (vr)
    redirect all incoming rigid edges to vr
    if edge(vr, vi) is the Longest_Path(vr, vi) then
        Edge redirection and constraint validation(CG, vi, vr)
    else
        return no solution found
    update the stack based on the CG
```

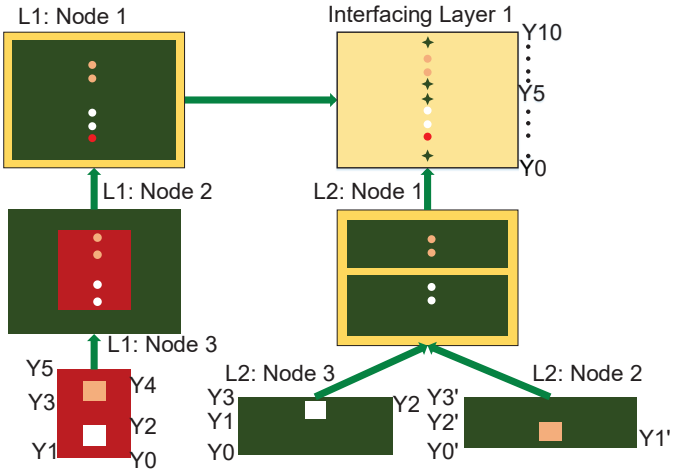

Fig. 7: Sub-tree structure of interfacing layer 1 in the Fig. 6(a)

location of this vertex can be instantaneously determined using the location of the corresponding reference vertex. Before marking a vertex as a removable candidate, all the incoming and outgoing edges need to be re-directed with respect to the reference vertex, which is performed by Algorithm 2. This algorithm checks if all incoming and outgoing edges can be referenced to or from the reference vertex of the dependent vertex with valid constraints. Upon satisfying the constraints and ensuring no positive cycle, all the edges are referred to the reference vertex by adding new edges, and the old edges are removed from the graph. After applying this algorithm, new rigid edges can be appeared and need to be processed. Therefore, the stack is updated in each iteration, and the process stops once all dependent vertices are processed. The time complexity of the process is linear with the number of dependent vertices. In Fig. 8(a) and (b), the backward edge is a negative weight, which ensures no positive cycle appears in the graph. After applying the algorithm, upon satisfying the criteria of rigid edge, all the incoming and outgoing edges are re-directed with respect to the corresponding reference vertex. Now, all the dependent vertices can be removed from the graph to reduce the graph size and improve the time complexity.

C. Hierarchical Constraint Propagation

To ensure children's constraints are compatible with the parent, we perform constraint propagation from the leaf nodes towards the root node. For each pair of shared vertices between parent and child, the longest path is propagated. The basic algorithm can be found in the previous works. Though that algorithm is for 2D/2.5D layouts, the hierarchical propagation concept is the same. The additional step that

Algorithm 2: Edge redirection and constraint validation

Input : CG, vi (dependent vertex), vr (reference vertex)
Output: updated CG

```
f = constraint(vr,vi)
foreach non-fixed incoming edge do
    vj = edge.source; e = edge.constraint
    if vj is a successor of vr then
        g = Longest_Path(vr,vj); weight = e-f
        if weight==g then
            add_edge(vr,vj,weight,rigid)
        else if weight<g then
            add_edge(vj,vr,weight,flexible)
        else
            return no solution found
    if vj is a predecessor of vr then
        h = Longest_Path(vj,vr); weight = max(h,e-f)
        add_edge(vj,vr,weight,flexible)
    remove the incoming edge
foreach outgoing edge do
    vj = edge.sink; e = edge.constraint; weight = f+e
    if vj is a predecessor of vr then
        h = Longest_Path(vj,vr)
        if weight+h==0 then
            add_edge(vr,vj,abs(weight),rigid)
        else
            add_edge(vr,vj,max(h,weight),flexible)
    else
        add_edge(vr,vj,weight,flexible)
    remove the outgoing edge
```

has been added in this work is vertex propagation. While propagating a fixed constraint, its reference vertex needs to be propagated as well. For example, in Fig. 7, there was initially no device footprint in the L1: Node 2. However, since the vertices from via have a dependency on the device vertex, it is propagated along with the vertices of vias from the child (L1:Node 2) to the parent (L1:Node 1) and upwards (interfacing layer 1). After propagating all necessary constraints and vertices throughout the sub-tree, the initial VCG of the interfacing layer is shown in Fig. 8 (a). Here, E2 (trace to device enclosure), S1 (trace to trace spacing), and E1 (substrate to trace enclosure) are propagated from the other intermediate nodes in the tree hierarchy. For the propagated edges in the initial VCG of the interfacing layer, each edge weight has the corresponding source layer id with it. E1 does not have the id as it can be found in both layers. Upon propagation, there are rigid edges, but the vertices are not dependent yet in the initial VCG. Therefore, the constraint validation and graph reduction algorithm (shown in Algorithm 1) has been applied to get the modified graph (shown at the top). Once all the necessary constraints are propagated to the root node, the graphs are ready for evaluation and layout generation.

D. Layout Generation and Optimization

1) Layout Generation: For generating layout solutions, the HCGs and VCGs are evaluated using the longest path algorithm. The layout generation workflow can be operated in three modes. To maximize the power density, a minim-sized solution can be generated by using the minimum constraints only. In this Mode 0, the floorplan size is determined from the root node's longest distance between source and sink vertices. To explore large variations in the solution layouts, the variable floorplan sizes layout generation approach can be pursued in Mode 1. In this mode, the edge weights of the root nodes are

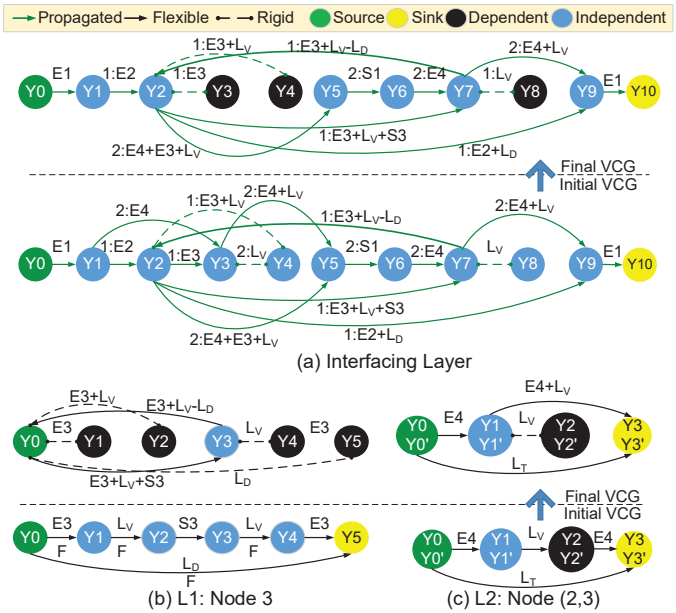

Fig. 8: Bottom-up constraint propagation for the structure shown in Fig. 7: (a) VCG of the interfacing layer, and (b) VCG of L1: Node 3 (left), L2 (Node 2, 3) (right).

Algorithm 3: Layout generation flow

Input : HCG, VCG, Layout hierarchy, User constraints
Output: Locations of the graph vertices for solution layout

```
Evaluate the root node using the longest path algorithm
while preorder travesal(Layout hierarchy) do
    propagate vertex locations from the parent
    while there are remaining edges do
        trim the graph and remove redundant vertex & edges
        partition the graph into independent subgraphs
        foreach subgraph do
            Evaluate locations(subgraph)
```

varied arbitrarily within a range, where the minimum constraint is the lower bound and the upper bound can be determined by the standard deviation provided from the user or some multiples of the minimum constraint value. So, the larger the standard deviation, the larger the variation can be generated in the layouts. Since power modules are used in power converters, in most cases, the user has a pre-defined floorplan size, for which the module layout needs to be optimized. Therefore, to support fixed floorplan size module optimization, the algorithms need to run in Mode 2, where the root node's source and sink vertex locations are defined by the user. If the floorplan size is larger or equal to the minimum floorplan size, then the methodology can generate an arbitrary number of layout solutions by randomizing the edge weights within the imposed limits due to the fixed size. The overall workflow is shown in Algorithm 3.

2) Layout Optimization Flow: The generated solutions are evaluated using the performance models. Optimization capability has been demonstrated using a 3D wire bondless module. In this work, both the native randomization and non-dominated sorting genetic algorithm (NSGAII) [14] have been used to perform electro-thermal optimization. Randomization is the built-in solution generation methodology for the tool, which exhaustively searches the solution space and can generate an arbitrary number of solutions. In randomization, both uniform distribution and truncated normal distribution functions can be used to generate new solutions. The genetic algorithm workflow

Algorithm 4: Evaluate locations

Input : CG
Output: Locations for the vertices

```
find the longest path
select the location evaluation candidate vertices
push all the candidates to a stack
make list of sources and sinks
while stack is not empty do
    current vertex (v) = stack.pop()
    min locations = longest path of v from each source
    max locations = longest path of each sink from v
    low = max(min locations); up = min(max locations)
    max limit = ((up-low)/length(stack))+low
    determine location of v = randomization(low,max limit)
    append current vertex to the sources and sinks list
```

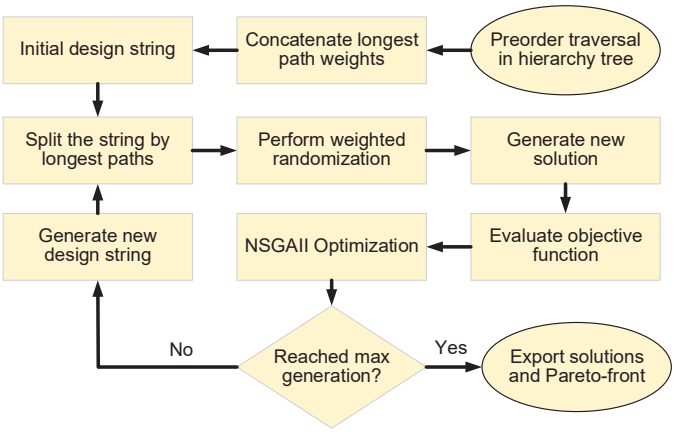

Fig. 9: NSGAII implementation workflow

is summarized in Fig. 9 with user-tunable configurations. Since randomization is the key to the layout generation workflow, for each longest path, the extra room is distributed among the edges according to the design string generated by the optimizer. From a study reported in [11], for 2D/2.5D layouts, the genetic algorithm can reach the Pareto-front faster than the randomization method. However, with a longer runtime, randomization can potentially find better solutions than the genetic algorithm. Both methods are used in this work to study the 3D layout optimization strategy.

IV. Results

To demonstrate the capability of the tool, three sets of solutions are generated. The tool has generic algorithms to generate 2D, 2.5D, 3D layout solutions. Minimum-sized solutions are generated for three sample cases (one of each kind). The solution layouts are shown in Fig. 10. From the results, it is clear that the 3D module has a much smaller footprint compared to a 2D and 2.5D module. With the same number of devices, the power loop inductance is found only 2.11 nH for the 3D module, whereas the 2D/2.5D module gives a loop inductance of 7.12 nH. Due to a smaller footprint, the thermal results could be worse in the 3D module. However, this face-to-back configuration allows double-sided cooling, which gives better thermal measurement for the 3D one compared to the 2D.

To optimize a 3D MCPM, a wire bondless module with three devices per switching position is selected. The minimum solution has a floorplan size of 20.7 mm $\times$ 12.2 mm, and loop inductance, the temperature rise is 1.53 nH, 62.24 °C, respectively. For optimization, five floorplan sizes are selected: 26 mm $\times$ 18 mm, 29 mm $\times$ 19 mm, 32 mm $\times$ 20 mm, 35 mm $\times$ 21 mm, 38 mm $\times$ 22 mm. For

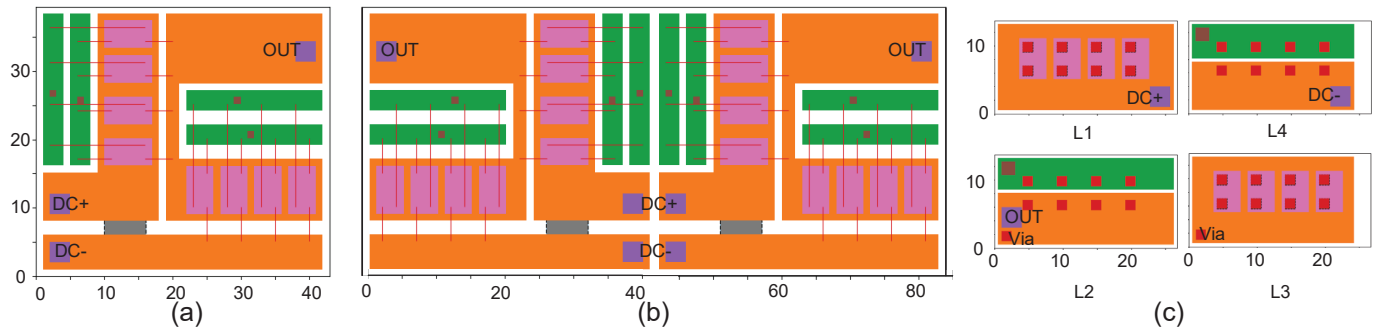

Fig. 10: Minimum-sized solutions: (a) 2D half-bridge power module (43 mm × 38.4 mm), (b) 2.5D full-bridge power module (84 mm × 37 mm), and (c) planar view of each layer of a 3D half-bridge power module (26.7 mm × 13.7 mm)

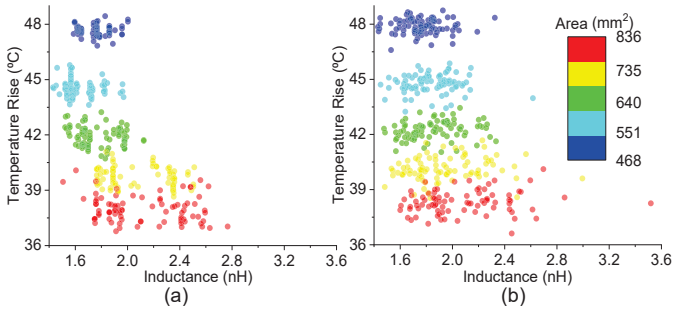

Fig. 11: Solution space with five different floorplan sizes (85 solutions/size): (a) NSGAII, (b) randomization

TABLE I: Runtime comparison between NSGAII vs Randomization

Algorithm	Total layouts	On Pareto-front	Approximate runtime (min)	
			Generation	Evaluation
NSGAII	937	148	25	206
Randomization	937	10	1	212

each floorplan size, 85 solutions are generated in the first iteration using both NSGAII, and randomization, and the solution spaces are shown in Fig. 11. Both of the solution spaces are showing a similar trend. Randomization is exploring more diversified solutions, including the similar quality solutions found by NSGAII. To find the Pareto-optimal solution space, NSGAII has been run for 100 generations and generated about 940 solutions for each size. The same number of solutions are generated by randomization, and the solution space is shown in Fig. 12(a). Runtime is computed using an Intel Xeon Silver 4210 2x10C@2.2G processor. The comparison result between NSGAII and randomization for a sample floorplan size solution set is shown in Table I. Though NSGAII cannot find better solutions, the number of solutions on the Pareto-front is higher than randomization, which is reasonable as randomization has no guidance towards optimization. The layout generation runtime can be improved by optimizing the implementation and paralleling the evaluation. Fig. 12(b) shows the Pareto-front comparison between the two methods. From the comparison, it can be seen that randomization has found better-optimized solutions for this layout compared to the NSGAII. This is because the 3D layout has a smaller variation compared to 2D with a reduced footprint. Therefore, all suitable optimization algorithms should find similar solution space. However, randomization searches exhaustively in the solution space and can find better solutions for some cases. Three selected solutions on the Pareto-front are shown in Fig. 13. The performance values for the layouts are shown in Table II. From the solution layouts, it is clear that layout A has the smallest footprint that gives the lowest inductance with the highest temperature rise. On the other hand, layout C has

TABLE II: Performance metrics for three layouts on Pareto-front

Layout ID	Inductance (nH)	Temperature Rise (°C)	Size (mm × mm)
A	1.37	46.99	26 × 18
B	1.37	37.96	38 × 22
C	2.54	36.72	38 × 22

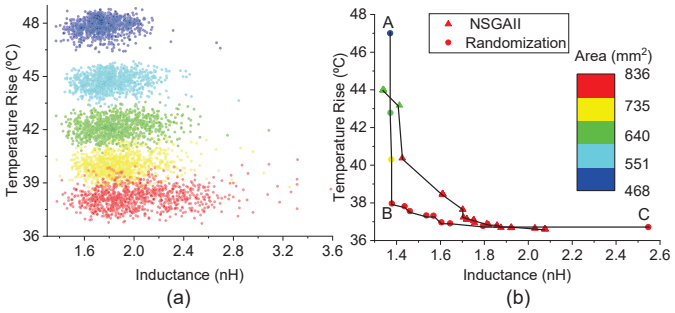

Fig. 12: (a) Randomization solution space with five floorplan sizes (945 solutions/size), (b) Pareto-front comparison with three selected layouts

the highest footprint with the devices spread out more evenly, which provides a higher inductance and lower temperature rise. In between two extreme cases, layout B has a balanced performance for both electrical and thermal.

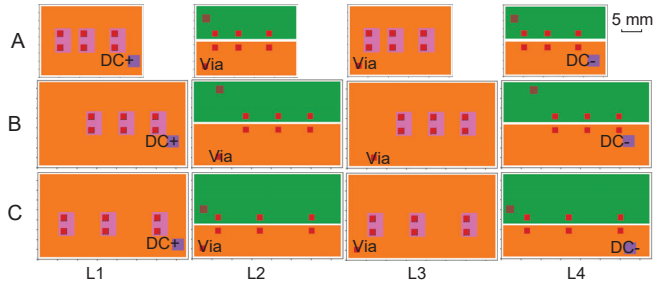

Fig. 13: Three selected layouts on the Pareto-front shown in Fig. 12

V. Conclusions and Future Works

We propose a power module layout synthesis and optimization framework promising for design automation in the power electronics industry. The capability to optimize all 2D/2.5D/3D power modules reaches state-of-the-art. The generic, scalable, and efficient algorithms can adapt to most existing packaging technologies in the industry. The current version relies on external tools and models, resulting in a relatively long runtime. They will be replaced by the built-in reduced-order models for accelerated runtime in future work, and our methodology and optimized modules will be hardware-validated through physical measurements.

REFERENCES

[1] X. Zhao, B. Gao, L. Zhang *et al.*, "Performance Optimization of A 1.2kV SiC High density Half Bridge Power Module in 3D Package," in *IEEE Applied Power Electronics Conference and Exposition*, Mar. 2018, pp. 1266–1271.

[2] C. Chen, F. Luo, and Y. Kang, "A Review of SiC Power Module Packaging: Layout, Material System and Integration," *CPSS Transactions on Power Electronics and Applications*, vol. 2, no. 3, pp. 170–186, 2017.

[3] P. Ning, H. Li, Y. Huang, and Y. Kang, "Review of Power Module Automatic Layout Optimization Methods in Electric Vehicle Applications," *Chinese Journal of Electrical Engineering*, vol. 6, no. 3, pp. 8–24, 2020.

[4] H. Chen, M. Liu, B. Xu *et al.*, "MAGICAL: An Open- Source Fully Automated Analog IC Layout System from Netlist to GDSII," *IEEE Design & Test*, vol. 38, no. 2, pp. 19–26, 2021.

[5] B. Prautsch, H. Dornelas, R. Wittmann *et al.*, "AnastASICA – Towards Structured and Automated Analog/Mixed-Signal IC Design for Automotive Electronics," in *ANALOG 2020: ITG/GMM-Symposium*, 2020, pp. 1–6.

[6] K. Kunal, M. Madhusudan, A. K. Sharma *et al.*, "INVITED: ALIGN – Open-Source Analog Layout Automation from the Ground Up," in *ACM Design Automation Conference*, 2019, pp. 1–4.

[7] T. Ajayi, V. A. Chhabria, M. Fogaça *et al.*, "Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project," in *ACM Design Automation Conference*, 2019.

[8] N. Hingora, X. Liu, Y. Feng *et al.*, "Power-CAD: A Novel Methodology for Design, Analysis and Optimization of Power Electronic Module layouts," in *IEEE Energy Conversion Congress and Expo*, Sept 2010, pp. 2692–2699.

[9] P. Ning, X. Wen, L. Li, and H. Cao, "An Improved Planar Module Automatic Layout Method for Large Number of Dies," *CES Transactions on Electrical Machines and Systems*, vol. 1, no. 4, pp. 411–417, Dec. 2017.

[10] T. Evans, Q. Le, S. Mukherjee *et al.*, "Powersynth: A Power Module Layout Generation Tool," *IEEE Transactions on Power Electronics*, vol. 34, no. 6, pp. 5063–5078, Jun. 2019.

[11] I. Al Razi, Q. M. Le, T. M. Evans *et al.*, "PowerSynth Design Automation Flow for Hierarchical and Heterogeneous 2.5D Multi-Chip Power Modules," *IEEE Transactions on Power Electronics*, vol. 36, no. 8, pp. 8919–8933, 2021.

[12] I. Al Razi, Q. Le, H. A. Mantooth, and Y. Peng, "Physical Design Automation for High-Density 3D Power Module Layout Synthesis and Optimization," in *IEEE Energy Conversion Congress and Exposition*, Oct. 2020, pp. 1984–1991.

[13] H. Chen, M. M. Hossain, D. Gonzalez Castillo *et al.*, "Design and Optimization of SiC MOSFET Wire Bondless Power Modules," in *IEEE International Power Electronics and Motion Control Conference*, 2020, pp. 725–728.

[14] K. D. et al., "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[15] "PowerSynth," (Last accessed August 13, 2021). [Online]. Available: https://e3da.csce.uark.edu/release/PowerSynth/

[16] M. Kamon, M. Tsuk, and J. White, "FASTHENRY: a multipole-accelerated 3-D inductance extraction program," *IEEE Transactions on Microwave Theory and Techniques*, vol. 42, no. 9, pp. 1750–1758, 1994.

[17] "ARL ParaPower," (Last accessed July 25, 2021). [Online]. Available: https://github.com/USArmyResearchLab/ParaPower

[18] J. K. Ousterhout, "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 3, no. 1, pp. 87–100, January 1984.