Placement and Routing for Power Module Layout

Tristan M. Evans
Electrical Engineering Department
University of Arkansas
Fayetteville, AR, USA
tmevans@uark.edu

Yarui Peng
Computer Science and
Computer Engineering Department
University of Arkansas
Fayetteville, AR, USA
yrpeng@uark.edu

H. Alan Mantooth
Electrical Engineering Department
University of Arkansas
Fayetteville, AR, USA
mantooth@uark.edu

***Abstract*— PowerSynth is a tool for the generation and multi-objective optimization of power module layouts. However, this generation is limited to variations of the initial layout sketched by a designer. In an effort to provide a deeper and more thorough exploration of the resulting design space, this paper proposes a placement and routing methodology adapted from VLSI techniques to provide a more diverse set of starting point layouts for PowerSynth optimization through synthesis of a circuit netlist. An overview of this method is presented along with prototype results of a continuous, force-directed placement routine. This paper demonstrates the proposed design flow, including PowerSynth initial layout results.**

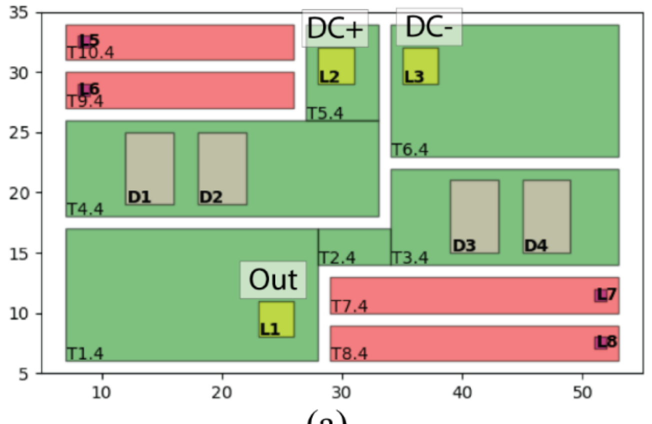
Keywords—power module, layout optimization, electronic design automation

I. Introduction

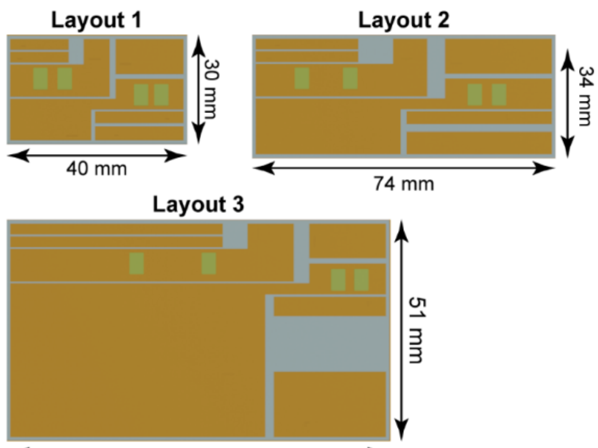
The design automation of power electronics modules is gaining steady traction in recent years with several tools and techniques being introduced to help aid the designer in exploring design space tradeoffs [1-4]. Among these tools, PowerSynth [5-7], uses a unique constraint-aware layout engine to rapidly generate feasible, manufacturable iterations of a power module design while optimizing for electrical parasitics, thermal performance, and mechanical reliability. The results of which are presented to the user on a Pareto frontier of tradeoffs in the design spaces chosen. However, while PowerSynth excels at quickly producing and evaluating layouts, these generated layouts are inherently variations of a single, starting point artwork as shown in Fig. 1 and Table I and adapted from [7].

In order to further expand the design space considered by PowerSynth, it is necessary to create a more diverse set of initial layouts for a given design. Additionally, the introduction of another layer of abstraction on top of the PowerSynth layout engine would allow greater flexibility of use for power systems designers in exploring system-level tradeoffs. For these reasons, this work proposes a modified place-and-route routine that synthesizes power module layouts from a circuit netlist and produces a plurality of starting point layouts immediately ready for further evaluation using PowerSynth.

In approaching issues related to power module layout synthesis, a good candidate field to draw inspiration from is that of VLSI (very large-scale integration) design automation. In this design flow, synthesis results in a gate-level netlist that is then combined with additional design files to specify the physical design. In general, layout partitioning breaks up the design into several smaller subsystems to be addressed individually. Floorplanning handles the arrangement of these subsystems within the layout while placement and routing steps are introduced to connect all of the subsystems and optimize for metrics like wirelength and timing.



(a)



(b)

Fig. 1. The initial layout for PowerSynth and resulting optimized layouts

This material is based on work supported by The National Science Foundation under Grant No. EEC-1449548. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation

TABLE I. LAYOUT PERFORMANCE METRICS

	Dimensions (mm)	Inductance (nH)	R_{TH} ($Wm^{-1}K^{-1}$)	Stress (MPa)
Layout 1	50x30	9.93	0.204	556
Layout 2	84x34	7.23	0.206	704
Layout 3	106x61	9.26	0.203	816

When it comes to power electronics, a similar vein of thinking can be applied where converters and systems are comprised of power electronics building blocks that can be optimized individually before integration. These building blocks can also take the form of power modules that are comprised of varying arrangements of switching cells to achieve different topologies and current ratings. So, to further the design automation of power electronic modules and systems, the methods and algorithms from the mature field of VLSI design make great candidates for adaptation. A comparison of some aspects of VLSI and power module physical design has been adapted from [3] and presented in Table II. One fortunate conclusion that can be drawn from this is that since the number of devices and types of topologies employed in power modules is so much less, this can help to constrain aspects related to placement and routing in the early stages of this work.

TABLE II. VLSI VS. POWERMODULE LAYOUT ASPECTS

	VLSI Layout	Power Module Layout
Component Count	High (up to billions)	Low (generally <20 [3])
Component Dimensions	Regular	Irregular
Routing Layers	Multiple	Single (generally, with multi-layer and 3D primarily in academic works)
Primary Measurements	Footprint, timing delay	Footprint, electrical parasitics, junction temperature, mechanical stress

II. PLACE-AND-ROUTE ROUTINE

A. Overview

The main purpose of this work is to develop methods and algorithms necessary to automate the steps in synthesizing a power module design from a circuit netlist to an initial layout for the EDA tool PowerSynth. This is accomplished by adapting VLSI techniques such as force-directed placement along with a simple grid-based routing algorithm.

The main steps associated with these methods involve the placement of terminals and devices relative to them before routing the power and signal traces then determining bondwire locations. An illustration of the place-and-route routine is shown in Fig. 2. A user-supplied annotated circuit netlist is used to initialize the process by not only establishing the connectivity of components relative to terminals, but also specifying the module footprint and desired terminal locations. The netlist follows the same general format of an LTSpice-compatible netlist but with a few exceptions. Chiefly, the comment token is used to flag keywords used by this routine that include module footprint dimensions, units, and terminal names and locations. An example of this annotated netlist is shown in Fig. 3. The reasons behind providing module footprint and terminal locations in this netlist are two-fold. First, using the terminals as fixed points

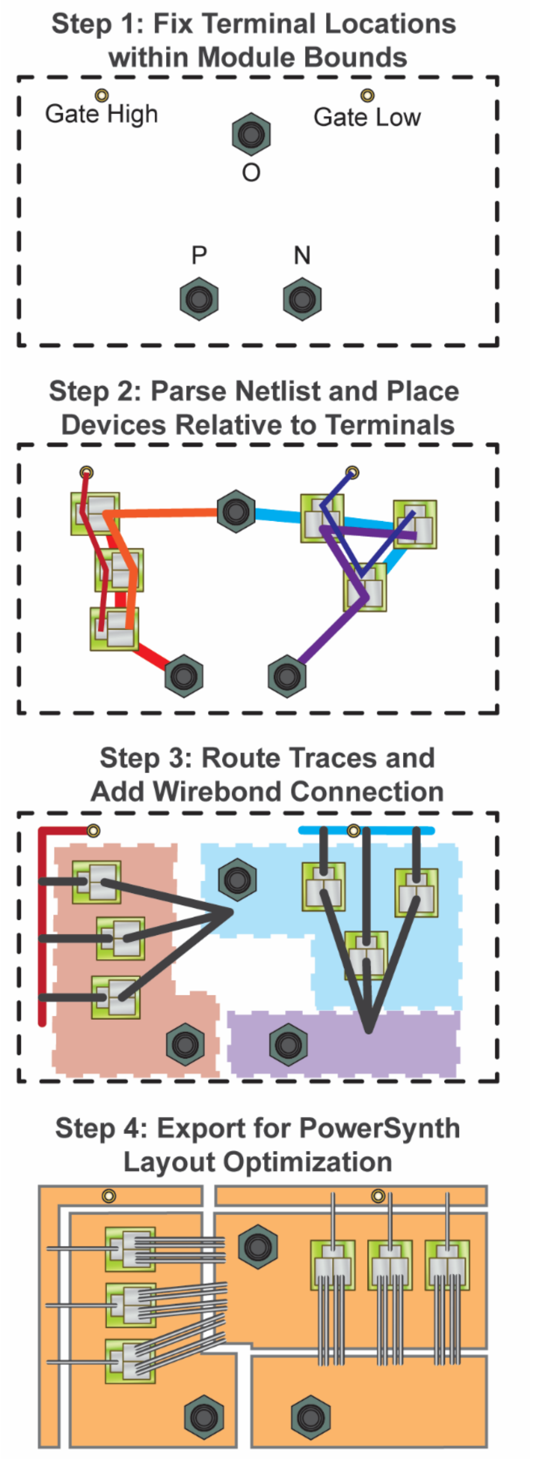
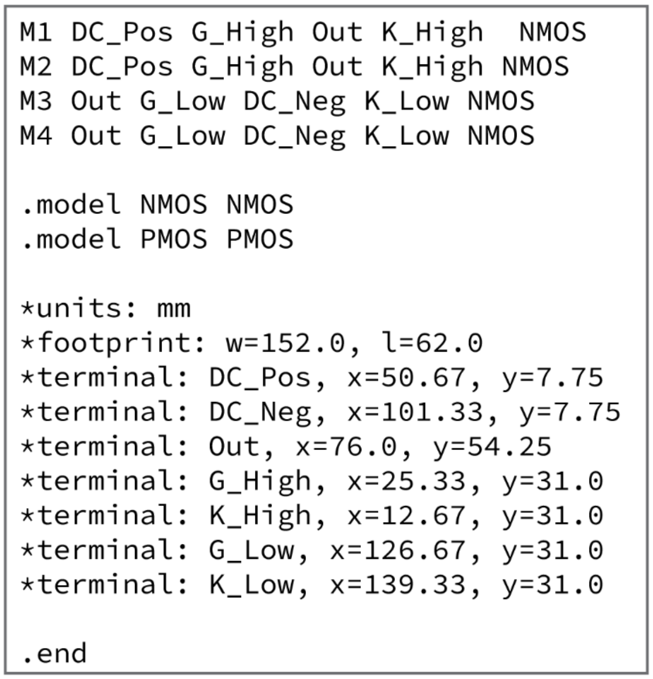

Step 1: Fix Terminal Locations within Module Bounds
Gate High
Gate Low
O
P
N
Step 2: Parse Netlist and Place Devices Relative to Terminals
Step 3: Route Traces and Add Wirebond Connection
Step 4: Export for PowerSynth Layout Optimization

Fig. 2. Place-and-route overview.



```
M1 DC_Pos G_High Out K_High  NMOS
M2 DC_Pos G_High Out K_High NMOS
M3 Out G_Low DC_Neg K_Low NMOS
M4 Out G_Low DC_Neg K_Low NMOS

.model NMOS NMOS
.model PMOS PMOS

*units: mm
*footprint: w=152.0, l=62.0
*terminal: DC_Pos, x=50.67, y=7.75
*terminal: DC_Neg, x=101.33, y=7.75
*terminal: Out, x=76.0, y=54.25
*terminal: G_High, x=25.33, y=31.0
*terminal: K_High, x=12.67, y=31.0
*terminal: G_Low, x=126.67, y=31.0
*terminal: K_Low, x=139.33, y=31.0

.end
```

Fig. 3. Example annotated netlist for a half bridge module with two devices in parallel per switch position. Note the additional parameters specified by the lines preceded with an asterisk.

helps to constrain the layout problem. Second, this opens up the potential for a circuit designer to quickly come up with a custom power module layout that fits their system design requirements—such as busbar geometry and gate driver locations.

Once the netlist has been imported and the terminals placed within the module outline, the netlist is parsed and connected components identified. This information is then used to populate the layout with the required devices and apply spring connections among them and their respective terminals, as explained in II.B. Following that, the grid-based routing algorithm again uses the netlist information to ensure connectivity of each device with its respective nets. Finally, the layout can be exported to PowerSynth where trace geometries and component locations are varied to explore electrical, thermal, and mechanical design-space tradeoffs.

The following sections cover the two main aspects of this routine in detail. It should be noted that, while the methods themselves are somewhat naïve, these simple rules are part of a larger effort to develop a platform for developing and testing algorithms for power module layout synthesis. As such, this prototype tool—mainly developed in Python and JavaScript—is not complete and follows a few caveats. These are addressed in the following sections and summarized in Section IV.

B. Force-Directed Placement

Placement of the devices relative to their connected components and terminals is done by applying spring forces among them in a force-directed placement routine as in [8-10]. First, terminals are placed in fixed locations as specified in the annotated netlist. Next, components are initially placed in random locations within the module footprint. These components could include various types of transistors, diodes, or passive elements. However, for the purpose of this explanation, only vertical MOSFETs are considered with drain, gate, source, and Kelvin source connections.

Once the layout is populated with devices, spring forces are applied among them, between them and their respective terminals, and between them and the boundary walls of the footprint. In general, these spring forces are defined using the Hooke's law as noted in [11] and shown below:

$$F = -k\Delta s \tag{1}$$

where F is the force applied to each device, k is the spring constant, and Δs the Euclidean distance between two connected components minus the rest length of the spring connecting them.

Paralleled MOSFETs sharing the same drain trace are considered to be a device group. Within such a group, springs forces are attractive, such as with M1 and M2 in Fig. 4. Each device in this group is also attached with an attractive force between themselves and their connected terminals (e.g., T1). Among all device pairs from groups other than their own, repulsive springs are inserted, as with M1 and M3 in Fig. 4. Not shown in this figure are the boundary forces applied to each device that are applied between the device and the four walls of the layout. For repulsive forces, a modified version of the equation above is used where a coefficient c is introduced to give a logarithmic response where the force grows as the components approach each other as shown below:

$$F = -k\log\left(\frac{\Delta s}{c}\right) \tag{2}$$

There are several design variables that are randomly determined to provide flexibility in the resulting layouts. These include initial device positions, device mass, damping forces, spring coefficients, and spring rest lengths.

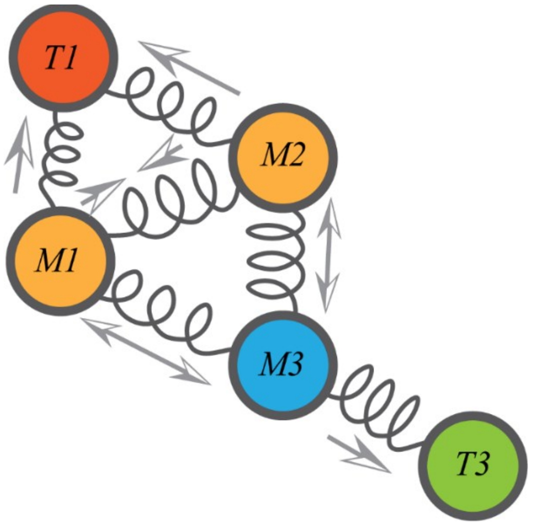


Fig. 4. Spring force illustration showing attractive (single-ended arrows) and repulsive (double-ended arrows) among layout components and terminals.

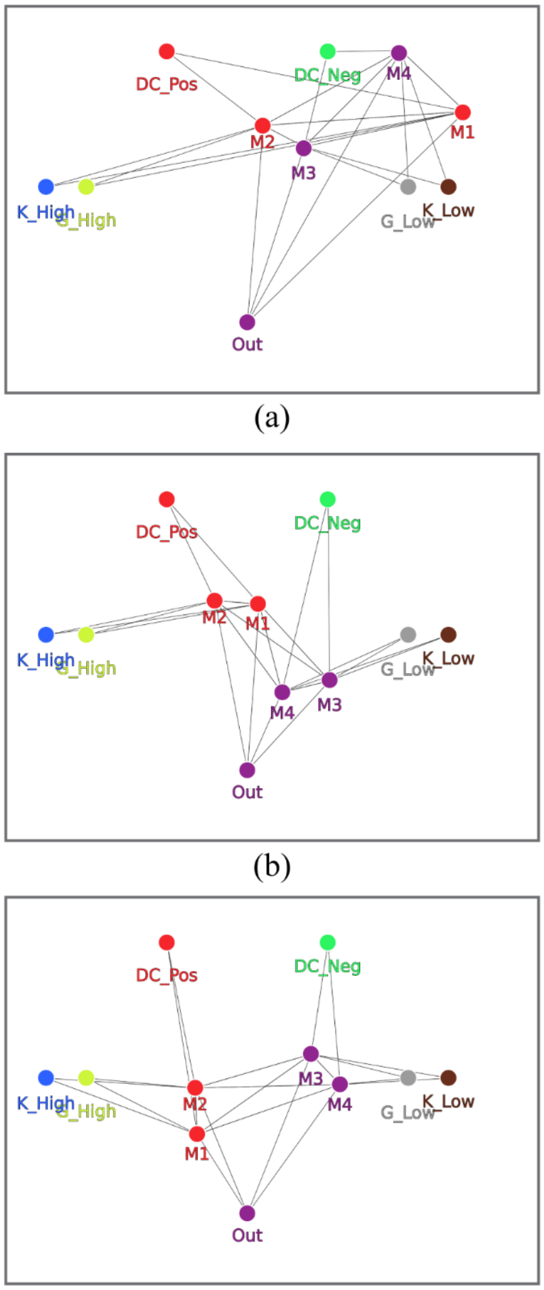


Fig. 5. Half-bridge component placement for layout with two devices per switch position. (a) Initial placement. (b) Intermediate results. (c) Final relative placement results ready for routing.

Solving this spring system is performed iteratively, in real-time, as the forces are updated and applied to each component to calculate their velocities and positions sixty times per second. This was chosen to ease development and debugging but has the downside of not being comparable in terms of run-time to other algorithms at this point. Despite this, the routine halts when the average system velocity approaches zero and generally completes within 10 seconds for simple designs. An example of several successive iterations of this process on a half-bridge module is shown in Fig. 5.

```
Data: Layout as a 2D grid with cells initialized for routing
Result: Power and signal trace routing solution for given layout initialization;
for $cell$ in $grid.cells$ do
    if $cell.locked = True$ then
        for $neighbor$ in $cell.neighbors$ do
            if $neighbor.group == None$ then
                neighbor.group = cell.group;
            else
                continue;
            end
        end
    end
    if $all\_cells\_locked()$ then
        loop = False;
    end
end
```

Fig. 6. Breadth-first search of grid cells to form traces.

C. Grid-Based Routing

Once the relative positioning of devices and terminals are established using the force-directed placement routine, a grid-based routing algorithm is used to create traces for all of the signal and power connections. Fundamentally, this routine relies on simple rules for assigning traces groups to the grid cells that depend on the state of their neighbors. This is accomplished using the breadth-first search routine in Fig. 6 and illustrated in Fig. 7.

A blank grid is sized proportionally to the layout with cells of user-defined resolution to begin this process. All of these cells are empty initially. During the initialization phase, the cells under each device and terminal are assigned a group designation reflecting the net they share in the annotated netlist. A cell that has a group designation is assigned the state of locked and will not be changed throughout this procedure. The routine begins by visiting each cell and pausing when it reaches a locked one. Then, from that locked cell, the neighboring cells in four directions are visited, as in Fig. 7 (a). Once a neighbor is visited, its group assignment is checked. If no assignment has been made, then the neighbor cell takes the state of the original cell and is locked. Otherwise, if the neighboring cell has a group assigned, it is ignored, and the algorithm continues. Once all neighbors of a locked cell are visited, then the next locked cell in the grid is visited, and the search continued. However, any neighboring cell that has just been assigned a group and locked will not be considered in this iteration of the routine. This ensures that cells from one group do not dominate the grid—as would be the case in a depth-first search.

One other important step occurs during cell initialization and prior to execution. This step draws a shortest line path between paralleled devices and the terminal they share for their drain connection using the netlist information. This is shown in Fig. 8 (a) where Bresenham's line algorithm [11] is used to assign the same group to cells between each paralleled device and their drain-side terminal. After that, the search algorithm continues as shown in Fig. 8 (b), and halts once all of the cell states are locked.

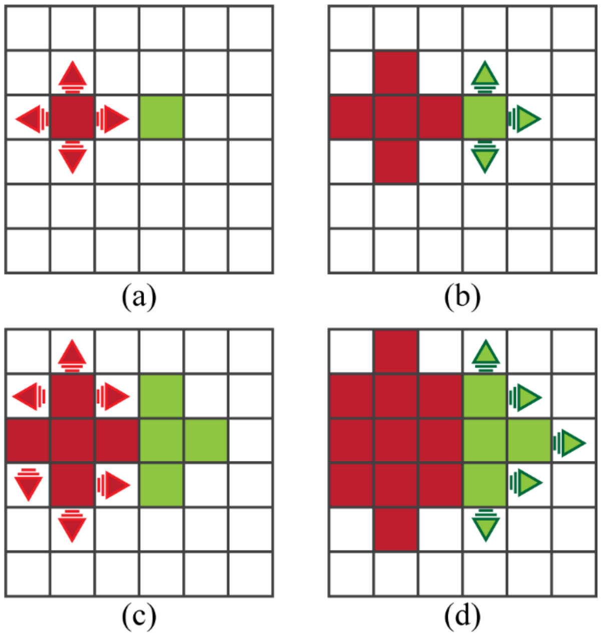


Fig. 7. Assigning group designations to cells based on initial starting conditions and neighboring cell states.

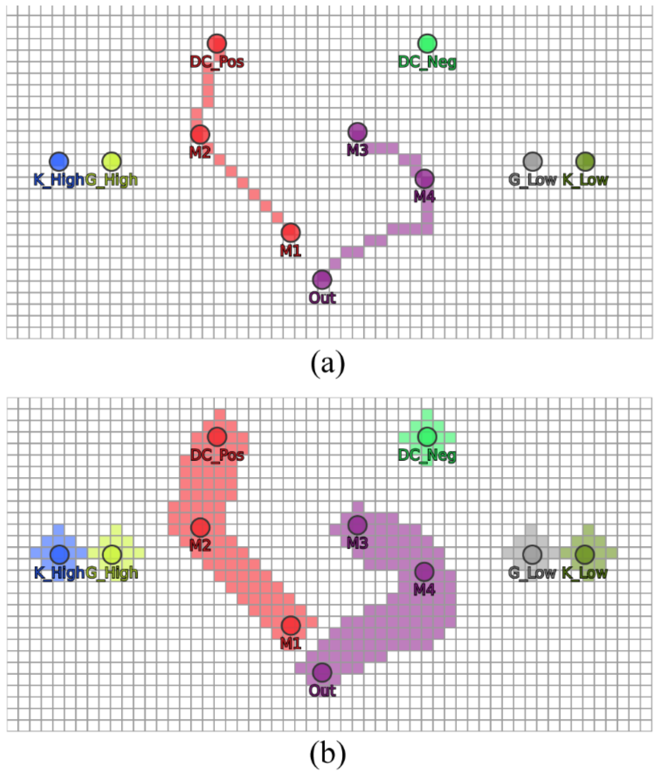


Fig. 8. Routing routine for the half bridge layout with (a) initialization phase using Bresenham's line algorithm and (b) results after a few iterations. Here each color represents a different trace group.

As a final step, a naïve solution for bondwire routing from each device pad to its respective traces has been implemented. This simply connects each device pad to its closest trace with a wire and returns the start and endpoint coordinates. While effective, a more sophisticated algorithm is necessary to prevent wirebond crossing and overlap.

Similar to the placement algorithm, this also runs in real-time at sixty frames per second, with one cell and its neighbors being visited per frame. The layout in Fig. 8 takes under two seconds to compute.

III. Results

The final place-and-route results for the half bridge module presented throughout this paper are shown in Fig. 9. The goal of this work is to eventually automate the export of such a layout as initial input for PowerSynth. However, a one-click solution for this is still in development. One of the many challenges associated with this is that while the place-and-route routine can produce designs with non-Manhattan trace geometries, the current version of PowerSynth cannot. Therefore, the final step of transferring the design to PowerSynth is done manually with results shown in Fig. 10 where the PowerSynth layout engine has generated a starting-point layout ready for optimization.

When using this proposed methodology, many variations of a design can be quickly created from the same netlist file. Fig. 11 (a) contains an example of this where two half-bridge layouts have been generated using this place-and-route routine from a half-bridge netlist with three devices per switch position. The resulting starting-point layouts generated by PowerSynth are shown in Fig. 11(b).

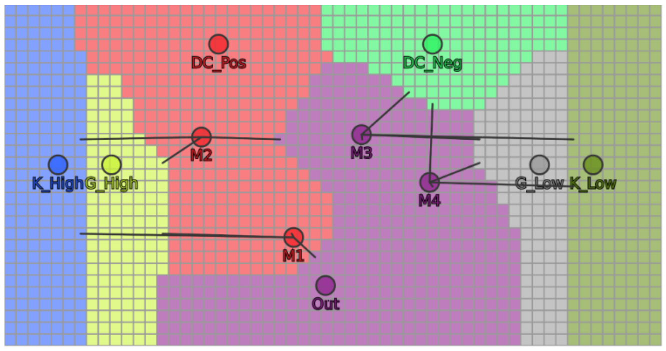


Fig. 9. Final placement and routing of the half-bridge module with wirebond connections.

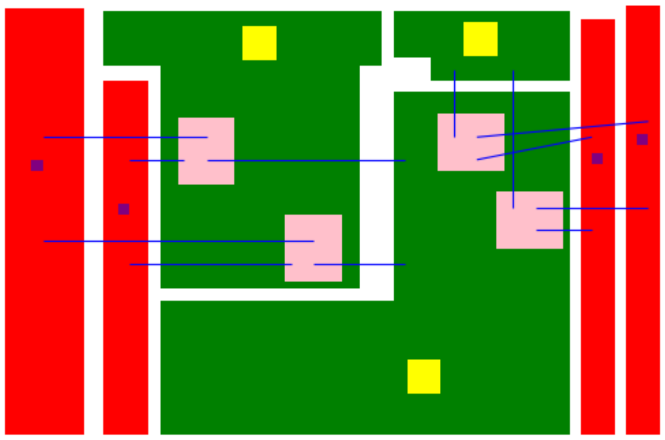
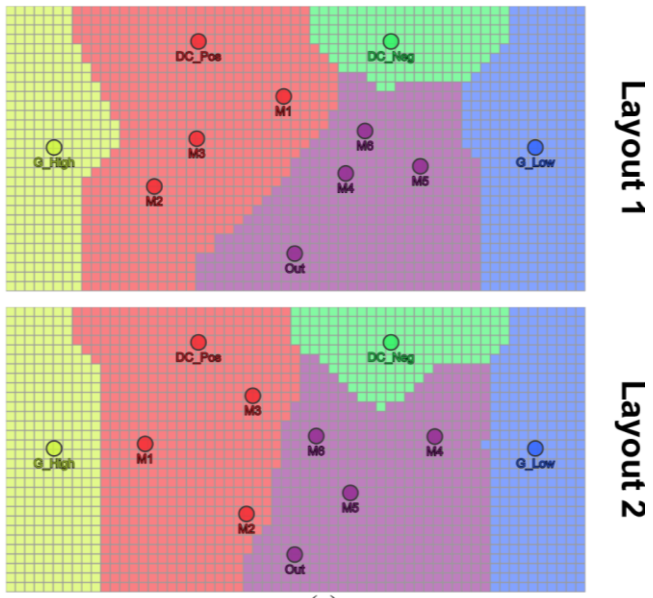
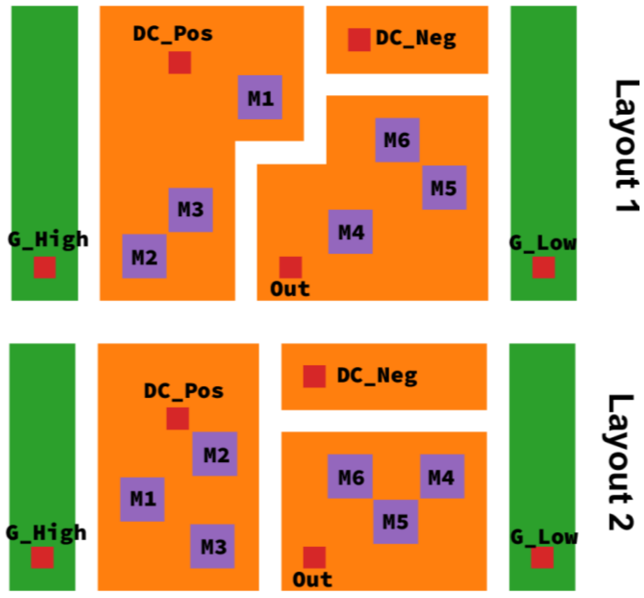


Fig. 10. Half-bridge layout generated by PowerSynth and based on the results in Fig. 9.



(a)



(b)

Fig. 11. Two layouts generated using the proposed place and route methodology (a) and their corresponding initial layouts in PowerSynth

IV. Conclusion and Future Work

By looking for inspiration in adjacent and more mature fields, this work explores the potential for using VLSI placement techniques with a grid-routing algorithm to automate the generation of power module layouts given an annotated netlist. While more work on these techniques is needed to introduce greater degrees of design freedom and robustness, this prototype demonstration shows a clear methodology that produces varying layout results with minimal user input in a short amount of time. As development progresses, more case studies will be performed using more nuanced techniques toward the goal of expanding the PowerSynth layout optimization design space.

Acknowledgment

The authors would like to thank Quang Le for help in testing and generating results for this paper.

References

[1] M. Hammadi, J. Y. Choley, O. Penas, J. Louati, A. Rivière, and M. Haddar. "Layout optimization of power modules using a sequentially coupled approach." *International Journal of Simulation Modelling*, 10(3):122-132, 2011.

[2] U. Drofenik, D. Cottet, A. Muesing, and J. W. Kolar. "Design Tools for Power Electronics : Trends and Innovations." *Ingénieurs de l'automobile*, 791:55-62, 2007.

[3] P. Ning, F. Wang, K. D. T. Ngo. "Automatic Layout Design for Power Module." *IEEE Transactions on Power Electronics*, 28(1):481-487, 2013.

[4] P. Ning, X. Wen, Y. Li, and X. Ge. "An improved automatic layout method for planar power module." In Conference Proceedings - *IEEE Applied Power Electronics Conference and Exposition - APEC*, volume 2016-May, pages 3080-3085, 2016.

[5] T. M. Evans, Q. Le, S. Mukherjee, I. Al Razi, T. Vrotsos, Y. Peng, and H. A. Mantooth. "PowerSynth: A power module layout generation tool." *IEEE Transactions on Power Electronics*, 34(6):5063-5078, 2019.

[6] I. Al Razi, Q. Le, H. A. Mantooth, and Y. Peng. "Constraint-Aware Algorithms for Heterogeneous Power Module Layout Synthesis and Optimization in PowerSynth." *IEEE Workshop on Wide Bandgap Power Devices and Applications*, pages 323-330, October 2018.

[7] T. Evans, S. Mukherjee, Y. Peng, and H. A. Mantooth. "Electronic Design Automation Tools and Considerations for Electro-Thermo-Mechanical Co-Design of High Voltage Power Modules." *IEEE Energy Conversion Congress and Exposition*, 2020.

[8] A. B. Kahng, J. Lienig, I. L. Markov, J. Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Springer, New York, 2011.

[9] C. J. Alpert, D. P. Mehta, S. S. Sapatnekar. *Handbook of Algorithms for Physical Design Automation*, CRC Press, Boca Raton. 2009.

[10] N. R. Quinn, M. A. Breuer, "A Forced Directed Component Placement Procedure for Printed Circuit Boards," *IEEE Transactions on Circuits and Systems*, 26(6):377-388 1979.

[11] J. E. Bresenham, "Algorithm for Computer Control of a Digital Plotter", *IBM Systems J.*, vol. 4, no. 1, pp. 25-30, Jan. 1965.