Automating GDS Generation in Magic

R. Timothy Edwards
SVP Analog and Platform
Efabless
San Jose, CA, USA
tim@efabless.com

***Abstract*—The Magic layout editor [1] has always been outside of the mainstream in EDA tools in its concept that custom manual layout should be simple and straightforward while remaining essentially unconstrained. Magic does this by hiding various implant layers, marker layers, and cut layers from the designer. As a consequence, it has to generate all these layers automatically during GDS format mask data generation. This is an extremely difficult problem. I will present a variety of examples requiring sophisticated decision-making on automated layout, strategies for producing DRC-correct layout, and a discussion of recent enhancements to Magic's GDS-generation engine and what further development work is required.**

***Index Terms*—EDA tools, open source, layout, VSLI**

I. Introduction

Automatic generation of custom analog circuit layout is a known difficult problem with a long history of research. Success comes mostly from modifying the nature of the problem, such that layout is arrayed, regular, and rules more easily satisfied by design, such as with digital placement and routing, or RAM compilers. Similar considerations for analog layout such as FaSoC [2] are promising. But as long as there is a need for full-custom analog layout, there is a benefit in simplifying the design process as much as possible, and automating layout generation in a way that removes the burden of many detailed layout requirements and places that burden on the software tools.

Many mask layers and design rules are the result of specific foundry details and add complexity to the layout process without any useful benefit. Such layers and rules force the designer to understand the nature of the design rules, and to run many iterations of layout drawing and DRC verification to identify, understand, and resolve issues.

The Magic layout editor has always sought to simplify the layout process for the designer, and makes use of many derived layers and automatically generated output [3]. Foundry process improvements outpaced software development for a long time, but recent development work helps reverse that trend.

II. Custom Layout Automation

Magic handles mask data output automation by providing a set of "recipes" in the technology file for a process that describes how to generate each GDS layer using a series of operators. Most of these operators are boolean logic operators such as OR, AND, and AND-NOT, and obvious geometrical operators like GROW and SHRINK. Unlike many EDA tools which have an object-based database, the plane-tesselating database magic makes some operations nearly trivial, such as a GROW operation followed by a SHRINK operation to bridge across narrow gaps in material. The tesselation method is shown in Fig. 1 and covers all space in a single plane with abutting rectangles, where areas not containing any material are represented by a type "space" which is similarly divided into simple abutting rectangles [4].

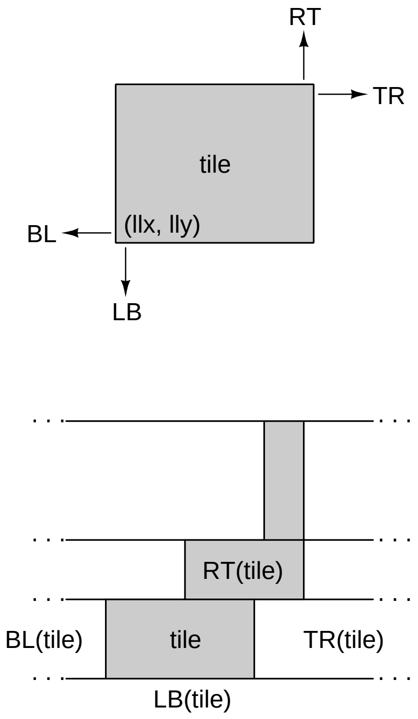

Fig. 1. Plane tesselation database method in Magic. Top: The tile structure with pointers to neighboring tiles. Bottom: A plane tesselated with material and space tiles.

The GROW operation is implemented by increasing the size of each tile of the specified type and painting into a new plane, which then replaces the original. The SHRINK operation is cleverly dependent on the full plane tesselation and can be implemented simply by increasing the size of each tile not of the specified type (including "space"), and painted into a new plane, which then replaces the original.

A set of straightforward operators was implemented in the earliest versions of magic. These are boolean operators and the GROW and SHRINK operators just mentioned. Also, a

SQUARES operator was defined so that Magic could describe contacts as “contact areas”, simplifying the creation of large-area contacts by not rendering cuts or representing them in the database, but generating them algorithmically on output. This is one example of how Magic simplifies the process of drawing layout, transferring the complexity to the mask data generation.

The small set of original operations implemented in Magic was completed by a context-dependent GROW operation called BLOAT-OR. The complete set of original operators was never sufficient even for early SCMOS processes, though, and as process feature sizes decreased, numerous gaps in the ability to automatically generate correct output mask data became apparent.

III. Hard Problems

The primary failure of the set of original operators is the handling of GROW followed by SHRINK for shapes positioned catecorner from one another. The situation is shown in Fig. 2, for a typical recipe to merge together implant areas that violate minimum spacing. If the shapes overlap slightly in the horizontal or vertical directions, then the recipe will leave behind a thin sliver of material that violates the layer’s minimum width rule. If the shapes don’t overlap, then nothing is generated, and the spacing violation remains [5].

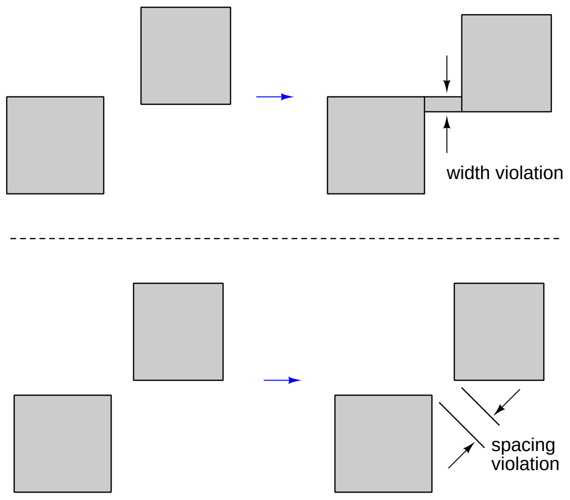

Fig. 2. Catecorner geometry causing issues with auto-filling spacing violations. Top: Geometry overlapping in the vertical direction. Bottom: Non-overlapping geometry.

Another difficult problem is the requirement for minimum width or area of an implant. In most cases, an implant such as N- or P-type doping just needs to surround an area of diffusion with some minimum overlap. But if the diffusion minimum width is small, then the diffusion width plus the implant overlap distance in both directions may not add up to the minimum width of the implant. In that case, the implant needs additional overlap to make up the minimum width, and no simple rule dictates in which direction (or both) the material should be expanded to make up the difference.

IV. Enhancements to Automatic Mask Data Generation

This section describes additional mask-generation operators that have been added to magic to resolve issues and gaps in the original set of operators.

A. *Contact cut arrays*

A new algorithm for contact cut generation was added which allows contact cuts to be non-rectangular. The algorithm first splits the contact area into maximum-size rectangles, and fills them with cut arrays, then picks up any remaining areas and fills them with cut arrays if there is sufficient room. Where maximum-size rectangles overlap, such as at the corner of an L-bend, the area of overlap should be handled first, and separately. This algorithm handles complex guard ring shapes, placing contacts at corners and intersections, and centering the remaining cuts along the ring, as demonstrated in Fig. 3.

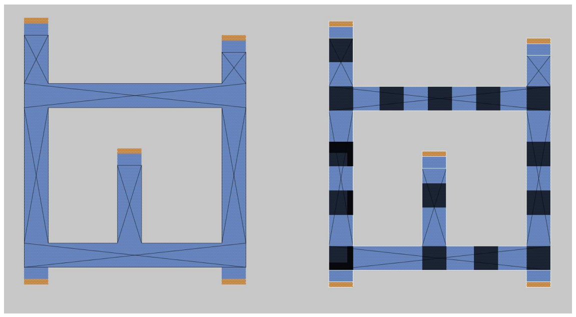

Fig. 3. Complex guard ring shape and automatic placement of contact cuts. Note the positioning of a contact at each intersection.

In addition to the new algorithm, a SLOTS operator has been added that extends the SQAURES operator to both non-square shapes, and shapes that may be placed at offsets instead of a regular grid. The offset shapes are well suited to the generation of fill shapes to meet layer density requirements.

B. *CLOSE operator*

A number of processes define a “minimum enclosed area” for layers, and much like spacing rules for implants, the preferred handling of violations is simply to add material to plug up the area of the violation. The CLOSE operator performs this task by identifying enclosed spaces of less than the required minimum, which it does by the simple method of summing the area of bounded “space” tiles. Having identified such a bounded area, it converts the “space” tiles to the implant layer type.

C. *BRIDGE operator*

The BRIDGE operator solves the catecorner problem in a deterministic way. The method of BRIDGE is to identify opposing tile corners, either corners of a specific layer type with less than the layer’s minimum distance between them, or corners of type “space” with less than the layer’s minimum width between them. In both cases, the area between the corners is filled with the material type in such a way as to satisfy all width and spacing rules (see Figs. 4 and 5). While the result is deterministic, it is not guaranteed to be the best solution, nor is it guaranteed not to create DRC violations with other material types.

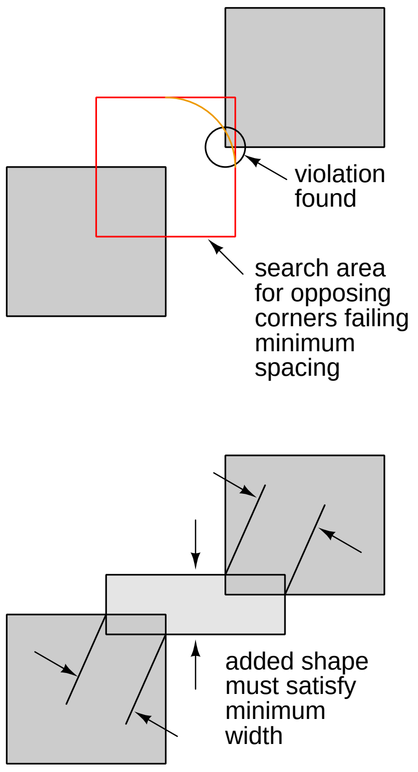
violation found
search area for opposing corners failing minimum spacing
added shape must satisfy minimum width

Fig. 4. Illustration of bridging algorithm.

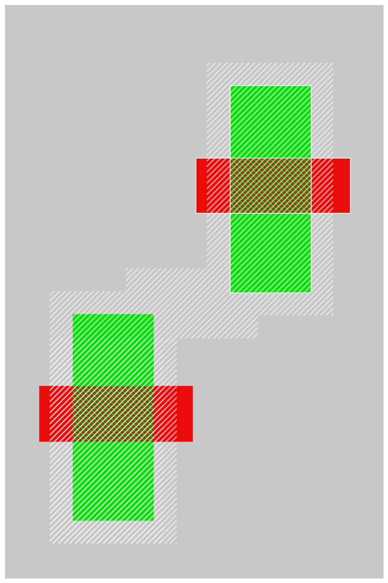

Fig. 5. Result of automatic bridging of catecorner shapes when generating N+ implant around N-diffusion (implant area shown in white crosshatching).

D. *BLOAT-ALL operator*

The layer-dependent BLOAT-OR operator fails to capture "interacting"-type rules in which, say, an implant must cover an entire layer based on context, such as when an nwell must be completely covered by a high-voltage implant if it contains high-voltage transistors. The BLOAT-ALL operator adds this functionality by growing a seed area (such as the area of a high voltage transistor) to completely fill the area of extent of another layer (such as an nwell).

E. *BBOX operator*

The BBOX operator is useful for certain rules that are dependent on the boundary of a cell. The most common use of this is the "bbox top" option, which restricts operation except on the topmost level of hierarchy in the layout. It is particularly useful for region identifiers at the top level, such as identifiers that indicate the padframe area for ESD, or the chip core area for latch-up rules.

F. *GROW-MIN operator*

The GROW-MIN operator is an attempted solution for the problem of layers requiring minimum width. The algorithm is to make a width check on every tile of the specified type, and if the minimum width rule is not met, then the tile is increased equally in both directions to make up the difference. This method is too trivially simple to handle complex geometry; it works well for transistor implants, where it only ever operates on simple rectangles (see Fig. 6).

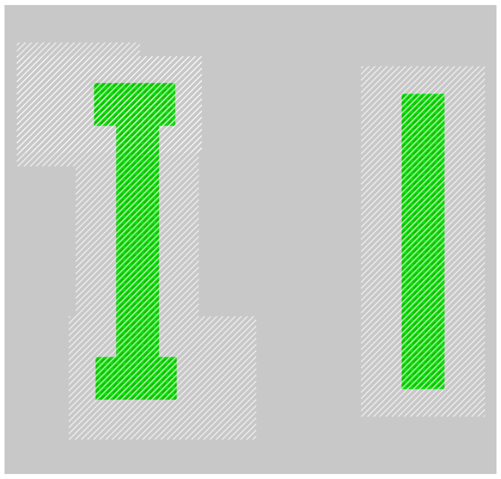

Fig. 6. The GROW-MIN operator works well on simple rectangles (right) but not so well on complex shapes (left).

G. *MASK-HINTS operator*

Ultimately, if there are situations where automatic mask generation fails to create DRC-clean layout, it must be possible to manually override the automatic generation. The MASK-HINTS operator method searches a cell for corresponding properties (dictionary key-value pairs) defining rectangular areas, and generates mask-data output corresponding to those areas (see Fig. 7). The drawback of this method is that if the cell layout is changed, then the property may become invalid without any indication. Although typically used to add material to a layout, the operator is always part of a recipe and can be used with the AND-NOT operator to define areas to erase as well as areas to add.

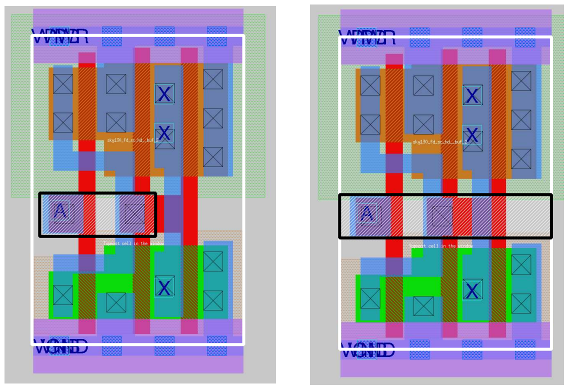

Fig. 7. A standard cell can lose mask information in Magic during GDS read. On the right side, a mask hint property is used to restore the original mask data, extending a contact implant layer (black outline) to reach both sides of the standard cell abutment box (white outline).

V. Areas Needing Improvement

Magic will resolve issues like bridging across corners and filling in gaps hierarchically from the top level of a layout downward. Typically, material is added at the topmost cell to resolve issues occurring between subcells. This method can run into trouble if a subcell alone causes automatically-generated layers to be generated differently than it would if the subcell geometry were flattened and merged with the parent cell. The GROW-MIN function can cause such issues because a subcell can contain a small amount of material requiring a surrounding automatically-generated implant to be expanded to meet a minimum width rule; but that subcell may abut more material in the parent cell, making the additional implant area unnecessary.

Ultimately, most issues are the result of a lack of feedback from the automatic mask data generation and the DRC engine in Magic. The BRIDGE operator, for instance, gets the layer width and spacing from the operator rule in the recipe; it does not perform an actual DRC check. Magic does have the ability to run DRC checks on the generated mask data planes; the concern with running them on GDS output would be the amount of time added to GDS layer generation. But in lieu of checks during GDS generation, it is necessary to ensure perfectly correct automatic layer generation. Since some methods like BRIDGE and GROW-MIN may have multiple possible solutions, some of which may result in DRC errors, these checks are probably a necessity.

A measure of feedback would also be useful with the SLOTS operator, to enhance its use for fill generation. The existing implementation of the SLOTS operator can apply a recipe with the hope of hitting the middle of a target range of pattern density. A preferable solution would be to specify a target density, and then have the SLOTS operation make adjustments to reach that target density.

VI. Conclusions

Complex foundry processes can be an impediment to quick and simple custom analog layout drawing. The open source layout tool Magic has an approach that simplifies the drawing process by automating much of the final mask generation through the use of geometric “recipes” that define the output layers. Process improvements have resulted in mask data requirements that Magic’s original set of operators cannot handle. An extended set of operators has been implemented to meet these demands. In a few hard cases, improved algorithms are needed to handle complex geometries. Ultimately, the methods need to be combined with feedback from the integrated DRC engine to ensure completely DRC-clean layout.

References

[1] R. Timothy Edwards, “Magic VLSI Layout Editor,” http://www.opencircuitdesign.com/magic

[2] T. Ajayi, et al., “An Open-source Framework for Autonomous SoC Design with Analog Block Generation,” 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SoC), Salt Lake City, USA, 2020

[3] J. Ousterhout, G. Hamachi, R. Mayo, W. Scott, and G. Taylor, “Magic: A VLSI layout system,” Berkeley internal document, December 3, 1982.

[4] J. Ousterhout, “Corner stitching: A data structuring technique for VLSI layout tools,” Berkeley internal document, December 13, 1982.

[5] R. Timothy Edwards, “The New Golden Age of Open Silicon,” Keynote presentation, Workshop on Open Source EDA Technology (WOSET) 2019.