Automatic Topology Optimization for Analog Module Generators

M. Wolf and U. Kleine

Otto-von-Guericke-Universität Magdeburg
IPE, PO Box 4120, D-39016 Magdeburg, Germany
email: mwolf@ipe.et.uni-magdeburg.de

Abstract

In this paper a new topology optimization feature of a module generator environment [5-6] will be presented. The optimization is performed by removing redundant elements of objects already placed and by assessing different layout topologies of a module. This drastically reduces the length of the generator source code, because different topologies need no separate source code, but result automatically.

1 Introduction

The quality and functionality of analog integrated circuits are strongly influenced by the layout of the circuit. In recent years, some tools for the automatic layout generation have been presented [1-4]. Often small layout variations can highly influence the quality of analog layouts. As a consequence, sub-optimal layouts can be the reason for rejecting automatic design tools by designers. The new optimization feature of a module generator environment [5-6] allows an automatic and controlled topology optimization by changing or removing redundant wiring and contacts. Due to the constructive and successive layout construction in this environment, the optimal layout topology can change during the module description. In order to avoid complex description with if/then statements for each topology, even objects already placed can be changed and optimized. This increases the reusability of modules, because modules, which have originally been written for another application and need a topology change for the new utilization, can be used without modification. The optimization feature keeps the source code small and easily readable.

2 Optimization Algorithm

The hierarchical data structure of the module generator environment forms the basis of the optimization algorithm. Modules are created in two steps in this environment: At first, geometric primitives are generated by a relative placement of objects using special functions. These geometric primitives and hierarchically built objects are then put together to complex modules by a special compactor. During module generation, not only the geometries are stored in the database but also the way of construction. Therefore, it is possible to rebuild an entire module under changed conditions. This feature will be used in the new optimization capability.

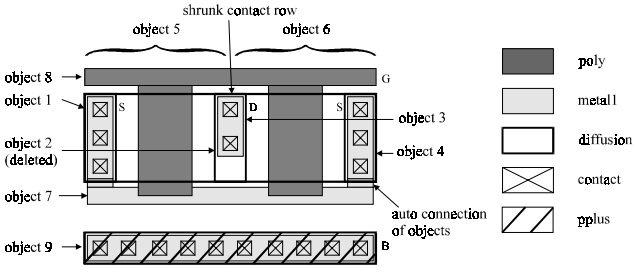

Fig. 1: Layout of the example

In Fig. 1, the layout of two parallel connected MOS transistors is depicted. With this hierarchical built example the optimization routine will be explained. During the entire module description, no design rules have to be regarded, this is automatically done by the environment. In this example two diffusion contact rows (obj. 1 and 2) have been compacted to a simple transistor and form object 5. Another transistor (obj. 6) has been created accordingly and compacted to the first transistor (obj. 5) deleting the redundant diffusion contact row (obj. 2). A metal1 rectangle (obj. 7) connects the source contacts and a poly rectangle (obj. 8) the gates. The substrate contact row (obj. 9) has been compacted to the bottom of the structure. The drain contact row has been shrunk to place the source connecting rectangle closer to the structure.

In this example the source and bulk terminal of the transistors are on different potentials. If these potentials are identical, the substrate contact row can be placed closer to the structure because the minimal distance between metal1 is no longer relevant. In a former version of the environment presented in [5], the objects are merged, but the metal1 rectangle, which has been inserted to connect the source contacts, is redundant and can be deleted. The source contacts can be connected by the metal1 rectangle of the substrate contact. If the metal1 rectangle is deleted, the parasitic capacitances and the crosstalk to the poly gates are decreased. Furthermore the metal1 rectangle of

the drain contact can be expanded again in order to decrease the contact resistance, because there is no more area improvement due to different design rules defining the distance of the substrate contact row to the transistor.

In Fig. 2a, the layout without optimization is illustrated. The metal1 rectangle of the substrate contact row has been connected with the metal1 rectangle of the source connector. In contrast to this, Fig. 2b shows the result with the deleted source connector after applying the new optimization feature.

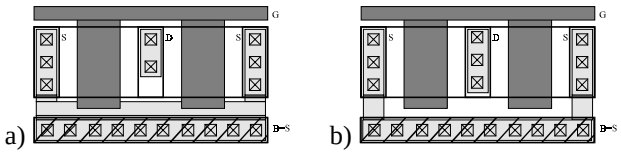

Fig. 2: Result without (a) and with optimization (b)

Besides the data structure for the construction of objects, each rectangle with its coordinates is also stored in the database. The coordinates are automatically calculated, the designer performs only a relative placement of objects. The algorithm for detecting redundant objects during a vertical compaction step is illustrated in the structogram of Fig. 3.

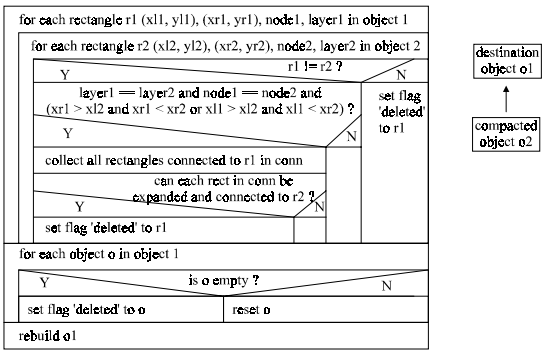

Fig. 3: Structogram for detecting redundant objects

Each rectangle r1 of the destination object o1 is compared with each rectangle r2 of the compacted object o2. Rectangles, which result from an array command, are not regarded as single rectangles in these loops. Only one surrounding rectangle is considered for an array in order to be able to delete even shifted arrays. If the two rectangles of the loops are identical (same layer, same potential, same geometry), the 'deleted' flag can be set directly. If the rectangles are not identical but overlap horizontally, all rectangles which are connected to r1 are collected and checked if they can be expanded upon r2. This check is necessary because after the deletion of r1 all rectangles, which had been connected to r1, must be connected to r2 in order to maintain the wiring of the layout. If all found rectangles can be expanded, rectangle r1 will be marked as deleted. Rectangles defining the shape of an array must overlap to be deleted.

The next step in the algorithm checks each object in the destination object if all rectangles have been deleted. In this case the object is marked as deleted.

The last step is to rebuild the entire object by a recursive function. The geometries of the entire object will be recalculated. This is necessary in order to cancel the modification of objects which have been done during the compaction of deleted objects for area optimization purposes. In the presented example the shrinking of the drain contact will be canceled. The rectangles, which had been connected to the deleted object, will automatically be connected to the new rectangle r2 by the auto connect feature of the compaction algorithm [5].

In Fig. 4 the layout of two compacted transistors with different sizes is depicted. In spite of the shifted contact rows, the redundant row in the middle has been deleted, because only the shape of a contact hole array is regarded.

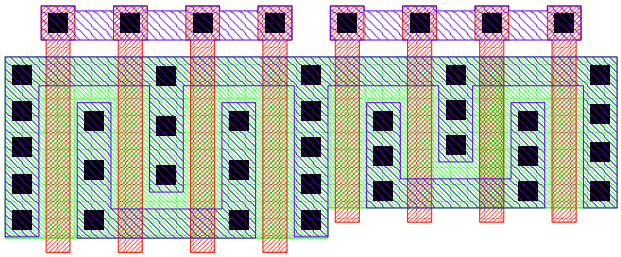

Fig. 4: Layout example for removed contact rows

The optimization algorithm also supports the detecting and deleting of contacts, which are unused. The designer can define additional contacts for the router, and redundant contacts will be deleted after the routing step.

3 Conclusion

In this paper a new optimization feature of a module generator has been presented, which automatically removes redundant objects of modules. The benefit is a more generic module description.

References

[1] J. Rijmenants, et al., "ILAC: An Automated Layout Tool for Analog CMOS Circuits," IEEE J. Solid-State Circuits, Vol. 24, No. 2, pp. 417-425, April 1989.

[2] H. Y. Koh, et al., "OPASYN: A Compiler for CMOS Operational Amplifiers," IEEE Trans. Computer-Aided Design, Vol. 9, No. 2, pp. 113-125, Feb. 1990.

[3] V. Meyer zu Bexten, et al., "ALSYN: Flexible Rule-Based Layout Synthesis for Analog IC's," IEEE J. Solid-State Circuits, Vol. 28, No. 3, pp. 261-268, March 1993.

[4] B. R. Owen, et al., "BALLISTIC: An Analog Layout Language," IEEE Custom Integrated Circuits Conference, pp. 3.5.1-3.5.4, 1995.

[5] M. Wolf, et al., "A Novel Analog Module Generator Environment," Proc. The European Design & Test Conference, pp. 388-392, March 1996.

[6] M. Wolf, et al., "Application Independent Module Generation in Analog Layouts," Proc. The European Design & Test Conference, p. 624, March 1997.