Resistance Extraction using a Routing Algorithm

Lorenz Ladage, Rainer Leupers

Universität Dortmund, Lehrstuhl Informatik 12
Postfach 50 05 00, D-4600 Dortmund 50, Germany

Abstract–This paper presents a new algorithm for calculating the resistance of an arbitrarily shaped polygon within a VLSI mask layout analysis program. In contrast to earlier approaches no polygon decomposition is required. Instead the current flow is determined by a routing algorithm. The resistance approximation is derived from the current flow. Experimental results have shown that this new algorithm achieves accurate results in comparatively little time.

I. Introduction

The extraction of parasitic resistances is one of the most important analysis steps in physical design. The main problem is to get the resistance value as accurate as possible within acceptable run time, especially for complex non-Manhattan style polygons.

The most accurate algorithms for resistance calculation solve Laplace's equations numerically ([1]). With respect to the complexity of VLSI design an approximate solution must be used. Recently published algorithms are based on the divide-and-conquer paradigm. An arbitrary polygon is decomposed into simple polygons. The resistance of a simple polygon is found by heuristics or taken from tables and the resulting resistance is just the sum over all simple polygon resistances ([4, 5]). Decomposition algorithms find resistance values within 10% of measured values ([5]). Since the resistance is an important factor for simulation, better results are desirable.

In general the value of the resistance R of a polygon can be calculated by the following equation:

$$R = R_S \cdot R_G$$

R_S denotes the sheet resistivity which depends on the specific resistivity and thickness of the conducting material. The factor R_G is called *geometric resistance*. R_G only depends on the shape of the polygon and the position of electrical nodes connected to it. R_S is intrinsic to the technology and will assumed to be constant for a given technology. In the special case of a rectangle we get:

$$R_G = \frac{L}{W}$$

where L is the distance along the current flow and W is the width perpendicular to the current flow.

The results of decomposition algorithms differ substantially, because of the difficulty to get an exact value for L. If L is determined heuristically for every simple polygon the fault accumulates. The new approach presented here comprises two sequential steps:

1. Get the *current flow line* S. S is the shortest path between two nodes in the polygon. The length of the *current flow line* S supplies a value for L because it represents the field of maximum current density. Thus we assume that the current flow can be identified by a single line.

2. Get the geometric resistance R_G by repeatedly probing the width of the polygon perpendicular to S. The equation $R_G = L/W$ is used in combination with an appropriate heuristic.

This approach is not restricted to Manhattan-style or 45° structures. The deviation of our results from measured values lies within 5% and the run time is suitable to use this approach in a VLSI mask layout environment.

The next section describes the system backround the extractor is part of. Then the routing part of the algorithm is described. The following section explains how the resistance is computed using the result of the routing part. The paper ends with experimental results and a conclusion.

II. System Overview

The resistance extractor is one feature of the CAMBIO analysis module. CAMBIO [2] is a CAD-system to support migration and optimization of geometric IC-layout. The process starts with an intensive layout analysis which will be explained in more detail later. During technology migration the input layout is modified to match the constraints of new technologies. The algorithm uses an one-dimensional constraint graph compaction technique. The module optimization is a post-processor after constructive layout manipulation tasks.

The optimization provides a set of useful heuristics to optimize layout with respect to chip area utilization.

The analysis module consists of the following parts:

1) Readers both for technology and layout data. CAMBIO supports all standard layout formats (CIF, GDSII and EDIF) and gets the technology information from the technology description language DINGO-XT.

2) Device recognition is performed by Boolean mask operations and topology relations like touch, cross and overlap.

3) The connectivity analysis delivers the electrical nets. It takes a set of polygons (after a separation of wires from gates) and identifies a net by pairwise overlap of polygons and contacts.

4) Parameter calculation consists of simple geometric operations (area or perimeter) and more complex algorithms like the resistance extraction which is the objective of this paper.

All extracted information is kept in the geometric symbolic data base, the central part of the CAMBIO system. In short terms the geometric symbolic data base keeps all links between corresponding structures in the three data base parts, technology, netlist and geometry. E. g. every geometric structure has a link to its corresponding device (netlist) and to the device type information (technology). The subsequent processing steps, migration and optimization, directly work on this data base. The combined geometric symbolic data base allows *device oriented* layout manipulation.

III. The Current Flow Line

Regarding the current flow in a polygon, it is assumed that most charge carriers move along the shortest connection between two contacts (the same consideration holds for other nodes like the source/drain of a MOS transistor). We call this connection the *current flow line* S. The Euclidean length of the *current flow line* therefore is a good approximation of L. The *current flow line* S in an arbitrary polygon will not be a straight line but will have some bends. So we may represent S by the path $P = (p_1, \ldots, p_n)$ (fig. 1).

For Manhattan metrics a lot of efficient algorithms exist for finding the shortest connection between two points in a polygon, which perform maze routing resp. line searching. ([6],[7],[3]). In the case of Euclidian metric the problem is quite difficult. Our algorithm uses a line-search approach, too, but looks for all possible and relevant solutions and stores them in a tree called path tree (function *BuildPathtree*). All these solutions are compressed (function *Compress*) and the best one is selected:

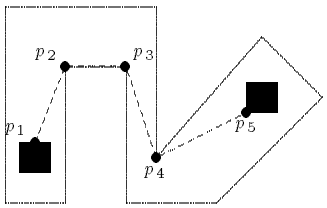

Figure 1: *Sketch of the current flow line. The current flow line is the shortest connection between two contacts that lies within the polygon.*

```
ShortestPath(startpoint, targetpoint)
{ BuildPathtree(pt);
  FOR EACH path w represented by pt DO
    Compress(w);
  OD
  Select the shortest path w*
  among the compressed paths;
}
```

The function *BuildPathtree* builds up a binary path tree whose nodes are points on the polygon or hole boundary which have been reached during the search. The function begins with the start point s. s becomes the root of the path tree. Starting from the root a try is made to reach the end point z. If an obstacle edge $e = (a, b)$ (either the polygon or a hole) is hit in point y, y will become the son of s in the path tree. a and b become sons of y (fig. 2). Then the search recursively starts from a and b until the end point is found. Since both possible paths around resp. along the obstacle are tried the end point is always found. If it is impossible to move directly towards the end point (i. e. point z in fig. 2) the obstacle is bypassed. Every obstacle causes two paths to be followed. Runtime can be considerably reduced by the following:

a) Every already reached point is stored in a hashtable together with its minimum distance to the start point. The tracking of path W is cancelled if the current point was already reached on a shorter path W'. W cannot lead to a better solution than W'.

b) That path is executed first which directs towards the end point. Tracking the other alternative may often be stopped because its total length grows beyond a boundary.

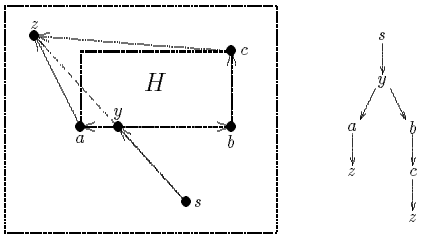

Figure 2: *Example for bypassing an obstacle. The direct line from s to z hits the hole H in point y. The search recursively starts from a and b. The hole H will be bypassed in both directions. The path tree (right figure) shows the result of the search. It represents the possible solution candidates* (s, y, a, z) *and* (s, y, b, c, z).

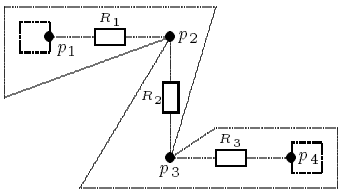

Figure 3: *Resistance network for S.*

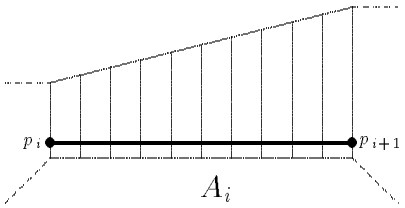

Figure 4: *Stripes of the cutting* A_i *with its segment* s_i.

The algorithm results in a complete path tree where every path from the root (s) to a leaf represents one possible solution. Because of the means mentioned above the number of possible solutions remains small.

A possible solution $P = (q_1, \ldots, q_m)$ normally contains some redundant points. A point q_i of the path is redundant if a direct connection exists between points q_{i-1} and q_{i+1} which does not hit an obstacle. (e. g. point y in fig. 2 is redundant on both paths). Redundant points are removed by the function *Compress* which tests for any triple (q_{i-1}, q_i, q_{i+1}) whether q_i is redundant. All compressed possible solutions are examined with regard to their length. The shortest path is selected. It becomes the *current flow line* $S = (p_1, \ldots, p_n)$. One can show:

1. The shortest path algorithm computes the shortest connection between any two points in an arbitrary polygon.

2. The average run time is $O(b \cdot n)$ where b is the number of bends in the optimal path and n is the number of polygon edges.

IV. Resistance Calculation

The Euclidian length of the *current flow line* S represents the value of L. In order to calculate the geometric resistance $R_G = L/W$ we still need the value of W. The value of W varies along S. If W is taken as an average over the whole *current flow line* this leads to inaccurate results. Therefore we look at W within the cuttings induced by $S = (p_1, \ldots, p_n)$: Between p_i and p_{i+1} the charge carriers move straight. This leads to the network in fig. 3, according to the given polygon and S.

The whole geometric resistance R_G of the polygon can be regarded as a serial network of part-resistances R_i

$$R_G = \sum_{i=1}^{n-1} R_i$$

where R_i is defined between the points p_i and p_{i+1} of S.

To obtain R_i we look at the cutting A_i caused by the segment $s_i := (p_i, p_{i+1})$. This cutting is separated into m stripes of equal length. Any stripe is perpendicular to s_i (fig. 4).

The geometric resistance of the cutting A_i may again be seen as a serial network

$$R_i = \sum_{j=1}^{m} R_{ij}$$

where each R_{ij} belongs to one stripe. The individual stripes are almost rectangles whose resistances obey the equation

$$R_{ij} = \frac{L_{ij}}{W_{ij}}$$

L_{ij} and W_{ij} are the length and width resp. of stripe j in cutting A_i (fig. 5). The values of L_{ij} are constant for fixed i. Note that the decomposition into stripes is not explicitly done but makes the explanation more transparent.

During experimental results the algorithm, in general, returned too small resistances because not all charge

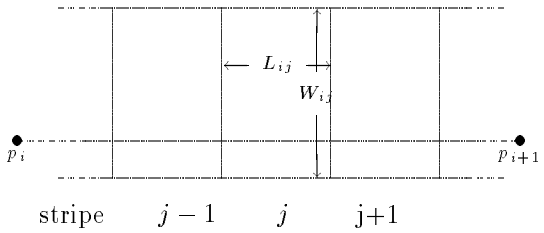

Figure 5: *Length and width of one stripe.*

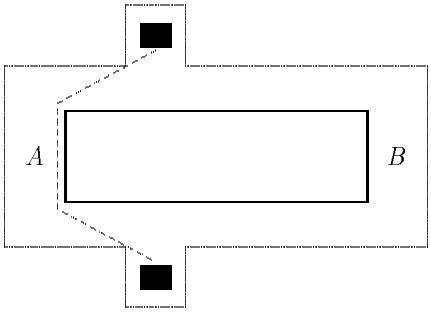

Figure 6: *Two branches A and B in a polygon with a hole.*

carriers move on the ideal line S. Some take a small deviation. This effect is removed by incorporating a small offset ΔR_{ij} for each stripe. ΔR_{ij} is proportional to the W/L ratio of every stripe. In turn this leads to small increase of L_{ij} by ΔL_{ij}. The more precise formula is:

$$L'_{ij} := L_{ij} + \underbrace{c \cdot \frac{W_{ij}}{|s_i|} \cdot L_{ij}}_{\Delta L_{ij}}, \qquad c \in [0,1]$$

where $|s_i|$ is the length of the segment $s_i = (p_i, p_{i+1})$. The parameter c controls the weighting of the W/L ratio by an increase of L_{ij}. Empirically a value of $c = 0.27$ turned out to be appropriate. The values of L'_{ij} may be different for all i, j because of the different values W_{ij}. The geometrical resistance R_G becomes the sum over all stripe resistances for all cuttings A_i:

$$R_G = \sum_{i=1}^{n-1} \sum_{j=1}^{m} \frac{L'_{ij}}{W_{ij}}$$

This method for resistance calculation in polygons may be extended to polygons with holes and to the case of more than two contacts:

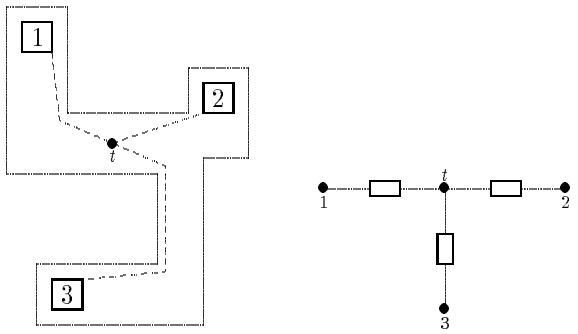

Figure 7: *Star shaped resistance network for more than two contacts.*

a) There are two alternatives to bypass a hole, e. g. the hole H of fig. 6 causes the two paths A and B. The routing algorithm of section 2 finds the indicated path. The resistance calculation of R_A is done as above. The branch A is then temporary blocked and the algorithm finds the *current flow line* in the branch B. R_B is calculated and the total resistance is computed for the resulting network. For this extension the holes have to be classified by their area. Relatively small holes should not be taken into account, because their contribution to the resistance can be neglected.

b) If the polygon contains more than two contacts our algorithm finds a resistance network. We introduce a settle point t for the polygon and find the shortest path from t to all contacts (fig. 7). Resistances are calculated as above and the polygon is replaced by a star shaped network. A suitable settle point $t = (t_x, t_y)$ for k contacts $c_1, \ldots, c_k$ is obtained by:

$$t_x := \frac{\sum_{i=1}^{k} c_k^x}{k} \qquad t_y := \frac{\sum_{i=1}^{k} c_k^y}{k}$$

where c_k^x and c_k^y are the x- resp. y-coordinates of the center of c_k. If t lies outside the polygon area it is moved to the nearest point within the polygon.

V. Experimental Results

The resistance extractor has been implemented in C++ on a SUN SparcStation. It has been tested on a wide range of individual polygons as well as on complete full

Table 1: Experimental results

Polygon	exact	extracted	error	CPU
1	9.16	9.81	7 %	0.34
2	2.60	2.66	2 %	0.14
3	2.20	2.44	10 %	0.08
4	6.80	6.82	0 %	0.37
5	12.05	11.43	5 %	0.32
6	10.70	10.40	3 %	0.49
7	4.40	4.44	1 %	0.13
8	6.90	6.70	3 %	0.31
9	10.40	10.71	3 %	0.59
10	4.75	4.87	3 %	0.31
11	4.00	3.62	10 %	0.29
12	7.35	6.93	6 %	0.48
13	6.85	6.70	2 %	0.40
14	8.50	8.53	0 %	0.18
15	18.00	17.80	1 %	0.43

custom layouts. Table 1 shows results for a selected set of polygons. Exact and extracted geometric resistances as well as accuracy and CPU time are given. The number of stripes (m) was set to 40.

The average accuracy was $g = 4.4\%$ for an average run time of 0.31 sec per polygon. Determination of the *current flow line* took about 0.1 sec. For Manhattan-style polygons a smaller value of m will achieve better run times without loss of accuracy. Another advantage is the small deviation of g that means that only rarely a result deviates more than 10% absolute from the exact value. Our algorithm achieves significant better accuracy (4.4% in contrast to 10%) than known decomposition techniques ([5],[4]) still having comparable run times. The resistance extractor ([1]) gets nearly exact results (within 3%) but needs about 1 min CPU time per polygon.

VI. Conclusion

A resistance extractor based on a routing algorithm is presented which overcomes the intrinsic error of decomposition methods because it tracks the flow of charge carriers. This leads to significant better results and a smaller absolute deviation from measured values. The average run time is suitable to use the algorithm in a VLSI environment. There are no restrictions to Manhattan or 45° structures, even polygons with holes may be handled.

References

[1] Barke, E.: Resistance Calculation from Mask Artwork Data by Finite Element Method, 22nd Design Automation Conference, 1985, pp. 305-311

[2] R. Brueck and K. Hahn and L. Ladage and E. Migas and B. Reusch and J. Waterkamp: CAMBIO: How to make cell libraries keep pace with technology evolution, FIP WG 10.5 Workshop, Grenoble, 1992, pp. 97–102

[3] Hightower, D.W.: A Solution to the Line-Routing Problem on the Continous Plane, 6th Design Automation Workshop, 1969, pp. 1-24

[4] Horowitz, M./Dutton, R.W.: Resistance Extraction from Mask Layout Data, IEEE Transactions on CAD, Vol. CAD-2, No. 3, 1983, pp. 145-150

[5] Hwang, J.P.: REX – A VLSI Parasitic Extraction Tool for Electromigration and Signal Analysis, 28th Design Automation Conference, 1991, pp. 717-722

[6] Lee, C.Y.: An Algorithm for Path Connections and its Applications, IRE Transactions on electronic Computers, Vol. EC-10, 1961, pp. 346-365

[7] Mikami, K./Tabuchi, K.: A Computer Program for Optimal Routing of Printed Circuit Connectors, IFIPS Proc., Vol. H47, 1968, pp. 1475-1478