Algorithm for Implementing Logical Operations on Sets of Orthogonal Polygons

Maxim Godovitsyn [1], Julia Zhivchikova [1], Nickolay Starostin [1] and Anton Shtanyuk [2]

[1] *Nizhny Novgorod State University n.a. N.I.Lobachevskiy, 23 Gagarin Avenue, Nizhny Novgorod, 603022, Russia*
[2] *Nizhny Novgorod Technical State University n.a. R.E. Alexeev, 24 Minin Street, Nizhny Novgorod, 603155, Russia*

Abstract

As part of the development CAD for design rule checks (DRC), it is necessary to use logical operations on orthogonal polygons that form the layout of an integrated circuit. Such operations as union, intersection, subtraction are performed over layers that contain orthogonal polygons. These operations are subject to stringent execution time requirements. The traditional representation of polygons in the form of bitmaps does not provide a quasi-linear dependence of time on the processed data size and requires development of new algorithms and polygon representation approaches. This paper contains a description of a modified sweeping line obscuring algorithm that achieves O(N log N) time. The algorithm uses three properties of the polygon: the separation of inner region from outer region by the edge, the belonging of edges to the set of either vertical or horizontal edges, and dissection of the layer plane into rectangular fragments which belong to either inner or outer region of the polygon. Procedures of input polygon contour representations that are dissected into sets of vertical and horizontal edges are described. As a result of performing logical operations, polygon edges of the resulting layer are formed. These edges, in turn, are converted into contour representations. The results of a computational experiment confirming the nature of the time dependences determined theoretically are presented. We propose the structure of a software system for DRC, built with the use of programming languages C++ and Lua.

Keywords

CAD, DRC, logical operations, orthogonal rectangles, polygons, sweeping line modified algorithm, IC

1. Introduction

The IC designing main task is obtaining a workable crystal layout, which will be used to create a template for fabrication. Before transferring the designed solutions to production, it is important to perform a cycle of obtained layout verification [1]. Verification is understood as a set of checks which successful passing provides the possibility of correct chip production on the particular manufacturing enterprise equipment and it's correspondence to the established parameters. Such checks include: design rule checking, layout versus schematic checking, electrical rule checking, parasitic extraction, etc.

The first stage of verification (DRC) consists in checking the layout for accordance with the design rules, which are described by the system of norms, restrictions, rules and procedures regulating permissible mutual arrangement of topological elements and topological structures, taking into account design features and possibilities of technological process.

GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27-30, 2021, Nizhny Novgorod, Russia
EMAIL: maxim.godovicyn@gmail.com (M. Godovitsyn); zhivchik96@mail.ru (J. Zhivchikova); nvstar@iani.unn.ru (N. Starostin); ashtanyuk@gmail.com (A. Shtanyuk)
ORCID: 0000-0001-8238-0665 (M. Godovitsyn); 0000-0003-4584-9414 (J. Zhivchikova); 0000-0003-1415-7511 (N. Starostin); 0000-0003-1809-7173 (A. Shtanyuk)

There are two phases in almost any verification procedure: the first one is the search of topological elements and specific areas, the last one is the check for design rules compliance directly.

Layout verification is usually performed using specialized CAD software. The most popular one is Calibre DRC [2,3], developed by the American company Mentor Graphics, purchased in 2017 by Siemens. There is a particularly acute question about the development of domestic CAD with similar functionality to Calibre DRC at present. Such program could solve several problems at once, including independence from foreign manufacturers, own distribution and licensing conditions. A domestic CAD program can comply with the Russian manufacturer's and customer's norms and rules for integrated circuits.

This paper focuses on describing the developed algorithms to perform logical operations (union, intersection, subtraction) with layers containing orthogonal polygons. These logical operations are actively used at the stage of search for topology elements and specific areas. Moreover procedures for device recognition in the IC layout are implemented on their basis [4]. The algorithms that implement logical operations are subject to strict requirements on computational costs, which in terms of time and memory should not exceed quasi-linear estimates.

2. Description of logical operations on the layers

We consider the integrated circuit layout, which is described by a set of orthogonal polygons. An orthogonal polygon is understood as a flat polygon with a boundary in the form of an orthogonal polyline, consisting of straight segments parallel to one of the axes of the crystal plane only [6]. The whole set of layout polygons is distributed in layers. A topological layer is understood as a set of pairwise non-intersecting orthogonal polygons. Any polygon key characteristic is belonging to a particular topological layer.

Depending on the presence of "holes" in the inner region of a polygon, it's boundary can be single-connected or multi-connected. Figure 1 shows examples of orthogonal polygons: a) simplest rectangles; b) polygon with a single connected boundary; c) polygon with a complex single connected boundary; d) polygon with a multi-connected boundary formed by "holes" in the polygon inner region.

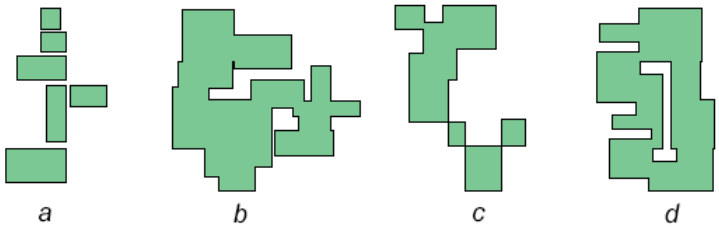

Figure 1: Orthogonal polygons examples

Conceptually, the verification procedures consist in recognition, search, separation of topology elements into specialized layers which various metrics are computed for and the obtained values are checked for compliance with norms and restrictions for. Any inconsistencies with the rules detected in the layers get into the verification reports, which serve as a basis for the subsequent error correction in the layout.

The basic mechanism for recognition, searching and selecting topology elements are the logical operations. The list of logical operations used to verify the norms of the design rules is given in Table 1. Each of the above operations takes two polygon layers as input. The result of a logical operation is a new layer containing a set of polygons. Figure 2 shows examples of the logical operations over two polygons result.

Table 1
Logical operations list for DRC

Operation	Name	Result
OR	Union	Area describing polygons related to the polygons of either the first and/or second layers.
AND	Intersection	Area describing polygons related to the polygons of the first and second layers simultaneously.
NOT	Subtraction	Polygons describing the polygon areas of the first layer, except for the overlapping areas of the second layer.
XOR	Exclusive OR	Polygons describing areas of the first or second layer polygon, except for the overlapping areas of the first and second layer.

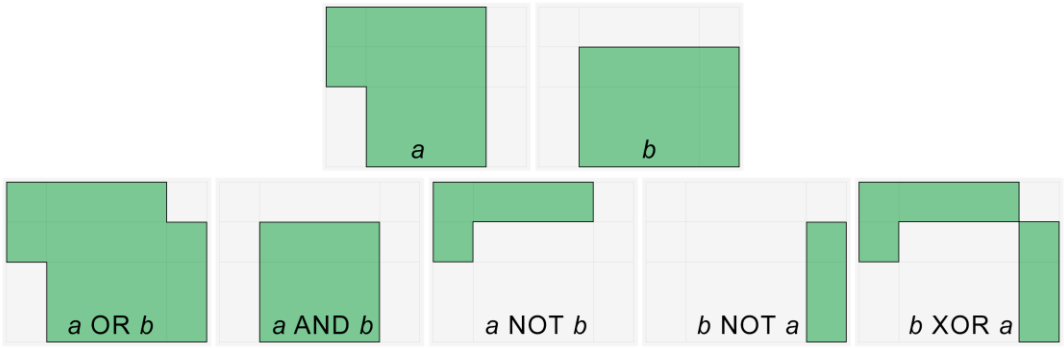

Figure 2: Examples of the logical operations with layers results

3. Contour representation and layer interior areas calculating procedure

It is proposed to describe the polygon by the coordinates of the polyline nodes (boundary nodes). Such a representation will be called contour representation, which is de facto standard representation for the layout at the verification stage [7]. In contour representation any topological layer is described by simple enumeration of all contours that form polygon boundaries. Each contour begins with an arbitrary starting node and ends with a finite node, which always coincides with the starting node. Such a representation is compact - the memory cost depends linearly on the number of nodes or edges of the layer contours.

Note that contour representation allows for variation in the input data representation: first, variation is possible in the choice of contour start nodes; second, any contour can be bypassed in two directions; third, variation is apparent in specifying the contour order in the enumeration. As an illustration, let us give an example of the polygon in Figure 3, which admits many variations of contour representation; for example, let us give two of them:

1: (0,2), (0,4), (4,4), (4,3), (1,3), (1,2), (2,2), (2,1), (4,1), (4,3), (5,3), (5,0), (1,0), (1,2), (0,2)
2: (0,2), (1,2), (1,0), (5,0), (5,3), (4,3), (4,1), (2,1), (2,2), (1,2), (1,3), (4,3), (4,4), (0,4), (0,2)

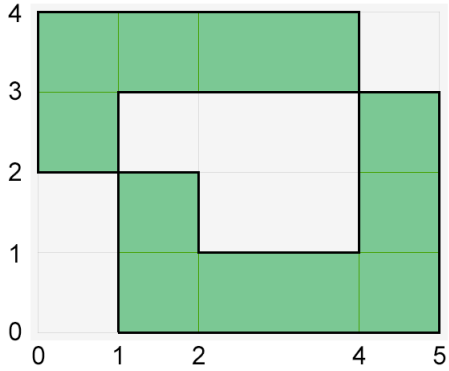

Figure 3: Example of a polygon with a complex boundary

Another problem of contour representation is that logical operations require information not about the boundaries, but about the inner regions of polygons. Thus, in view of all the above, using of contour representation in the algorithms for performing logical operations may be ineffective without solving these problems.

To resolve these issues, we will use the "sweeping line" scheme [5]. The idea is to exploit three properties of a polygon. First, any edge of a polygon boundary always separates the interior region of the polygon from the exterior. Second, any edge of an orthogonal polygon is either vertical or horizontal. Third, if the plane is dissected by straight lines according to the vertical and horizontal edges into rectangular fragments, then any such fragment will belong entirely to either the interior or the exterior region of the polygon.

Consider the horizontal edges of the polygon only. Arrange the edges in ascending (not descending) order of Y coordinate values, and form a queue. Then we extract from the queue the set of edges that have the same Y – such edges will be located on the same horizontal line. These edges of the form $(a, Y) - (b, Y)$ can be represented as half-intervals ($a \leq X < b$), where a and b are coordinate values along the X axis. By the definition of the layer, such half-intervals will never intersect even for the case of a polygon with a complex one-coherent boundary. As a result, on the horizontal line for a given value of Y we obtain segments of "switching" inner areas of the polygon to outer and vice versa outer to inner. It is enough to have a system of semi-intervals marking the inner regions of the polygon obtained at the previous value of Y to implement such “switching”.

Using the example shown in Figure 4, consider the workflow of the procedure for calculating the layer inner regions.

The initial data is an ordered set of horizontal edges H = ((1,0)-(5,0), (2,1)-(4,1), (0,2)-(1,2), (1,2)-(2,2), (1,3)-(4,3), (4,3)-(5,3), (0,4)-(4,4)).

Let us denote by I^{prev} – the system of intervals describing the polygon inner area immediately above the previous horizontal line. At the initial stage $I^{prev} = \emptyset$.

For Y=0 we extract a single edge (1,0)-(5,0) from H. Form B – a system of semi-intervals describing the boundaries for the current horizontal line: $B = \{[1,5)\}$. Calculate I – a system of intervals describing the inner area of the polygon directly above the current horizontal line by the following formula $I = B$ XOR I^{prev}. Here XOR – is an operation of exclusive OR over sets of intervals; the result of the operation is a new set containing those semi-intervals which belong to strictly one of the sets (either to the first or to the second one) only. For Y=0 we obtain $I = \{[1,5)\}$. $I^{prev} = I$ and proceeds to the next iteration.

For Y=1 we extract (2,1)-(4,1) from H. Form $B = \{[2,4)\}$. Compute $I = \{[2,4)\}$ XOR $\{[1,5)\}$ = $\{[1,2), [4,5)\}$. We take $I^{prev} = I$ and proceed to the next iteration.

For Y=2 we extract edges (0,2)-(1,2) and (1,2)-(2,2) from H. Form $B = \{[0,2)\}$. Compute I = $\{[1,2), [4,5)\}$ XOR $\{[0,2)\}$ = $\{[0,1), [4,5)\}$. We take $I^{prev} = I$ and proceed to the next iteration.

For Y=3 we extract edges (1,3)-(4,3) and (4,3)-(5,3) from H. Form $B = \{[1,5)\}$. Compute I = $\{[0,1), [4,5)\}$ XOR $\{[1,5)\}$ = $\{[0,4)\}$. We take $I^{prev} = I$ and proceed to the next iteration.

For Y=4 we extract edges (0,4)-(4,4) from H. Form $B = \{[0,4)\}$. Compute $I = \{[0,4)\}$ XOR $\{[0,4)\}$ =$\emptyset$. We take $I^{prev} = I$ and proceed to the next iteration.

The set H is empty, exit.

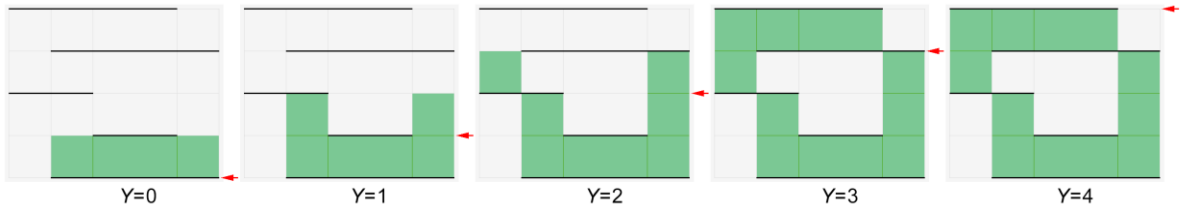

Figure 4: Schematic of the algorithm for calculating the layer's inner regions

Thus, the procedure for calculating the layer inner regions can be represented as the following scheme:

Procedure "A"

1. Initial data: H is an ordered set of horizontal edges;
2. Take $I^{prev} = \emptyset$;
3. If H is empty, then exit;
4. From H we extract edges with minimum Y value;
5. Form B – half-intervals describing the boundaries of the polygon for a given Y;
6. Compute $I = B$ XOR I^{prev};
7. Take $I^{prev} = I$;
8. Go to item 3.

If we take into account that we can use implementations of algorithms based on the classical interval tree [11] as functions providing work with half-intervals, then the computational complexity of this procedure in memory and in time is estimated as $O(N\ log\ N)$, where N – is the number of horizontal edges of layer polygons.

Returning to the problems of contour representation, we note that the presented scheme has the following features. First, it does not depend on the variability in the representation of initial data (not a contour, but a reordered set of edges is fed to the input). Second, it provides consistent computation of polygon regions with acceptable (quasi-linear) computational costs.

4. Generalized procedures for calculating the results of logical operations with layers

As noted above, the logical operation input is two layers described in the contour representation. The result of a logical operation is a new layer also described in the contour representation. The problem with contour representation is that the result of logical operations requires information not about the boundaries, but about the polygon inner regions. This problem has been solved using the procedure "A", presented in detail in the previous paragraph. Let us show how to construct on its basis a generalized algorithm for calculating the result of various logical operations with two layers.

The main idea consists in synchronous parallel calculation of the first and second layers inner regions on the basis of procedure "A". In doing so for each case of "sweeping" line displacement there are calculation of layer inner regions, execution of logical operation and calculation of the final layer inner regions, calculation of transitions (correspond to polygon boundaries) from inner regions to outer ones and vice versa from outer to inner ones, formation of polygon boundaries.

Here is the procedure "B" for calculating the result of a logical operation in the form of horizontal boundary edges of the final layer polygons.

Procedure «B»

1. Initial data:
   H_1 is an ordered set of horizontal edges of the layer 1;
   H_2 is an ordered set of horizontal edges of the layer 2;
2. Take:
   $I_1^{prev} = \emptyset$ – intervals of the inner region of layer 1 above the sweeping line;
   $I_2^{prev} = \emptyset$ – intervals of the inner region of layer 2 above the sweeping line;
   $I^{prev} = \emptyset$ – intervals of the inner region of resulting layer above the sweeping line;
   $H = \emptyset$ – horizontal edges of the resulting layer;
3. If $|H_1| + |H_2|$=0, then exit;
4. Let Y_1 – minimum Y for edges from H_1;
5. Let Y_2 – minimum Y for edges from H_2;
6. Compute $Y = \min\{Y_1, Y_2\}$;
7. If $Y = Y_1$, then
   a. Extract edges with the value Y from H_1 ;
   b. Form semi-intervals B_1;
   c. Compute $I_1 = B_1$ XOR I_1^{prev};
   d. Take I_1^{prev}=I_1;
8. If $Y = Y_2$, then
   a. Extract edges with the value Y from H_2 ;
   b. Form semi- intervals B_2;
   c. Compute $I_2 = B_2$ XOR I_2^{prev};
   d. Take I_2^{prev}=I_2;
9. Calculate the result of a logical operation $I = I_1^{prev}$ OPERATION I_2^{prev};
10. Compute $B = I$ XOR I^{prev} – the set of semi-intervals corresponding to the boundaries of the polygon-result of the logical operation;
11. Transform semi-intervals B into edges and include them in the set H.
12. Take $I^{prev} = I$;
13. Go to item 3.

Note that the result of Procedure "B" is the set H of horizontal edges of the resulting polygon. Let us give the procedure "C" for obtaining vertical edges, which is similar in its essence to the procedure for horizontal edges.

Procedure «C»

1. Initial data:
   V_1 is an ordered set of vertical edges of the layer 1;
   V_2 is an ordered set of vertical edges of the layer 2;
2. Take:
   $I_1^{prev} = \emptyset$ – intervals of the inner region of layer 1 to the right of the sweeping line;
   $I_2^{prev} = \emptyset$ – intervals of the inner region of layer 2 to the right of the sweeping line;
   $I^{prev} = \emptyset$ – intervals of the inner region of resulting layer to the right of the sweeping line
   $V = \emptyset$ – vertical edges of the resulting layer;
3. If $|V_1| + |V_2|$=0, then exit;
4. Let X_1 – min value X for edges from V_1;
5. Let X_2 – min value X for edges from V_2;
6. Compute $X = \min\{X_1, X_2\}$;
7. If $X = X_1$, then
   a. Extract edges with the value X from V_1 ;
   b. Form semi-intervals B_1;
   c. Compute $I_1 = B_1$ XOR I_1^{prev};
   d. Take I_1^{prev}=I_1;

8. If $X = X_2$, then
    a. Extract edges with the value X from V_2 ;
    b. Form semi-intervals B_2;
    c. Compute $I_2 = B_2$ XOR I_2^{prev};
    d. Take I_2^{prev}=I_2;
9. Calculate the result of a logical operation $I = I_1^{prev}$ OPERATION I_2^{prev};
10. Compute $B = I$ XOR I^{prev} – the set of semi-intervals corresponding to the boundaries of the polygon-result of the logical operation;
11. Transfer semi-intervals B into the edges and include them in the set V.
12. Take $I^{prev} = I$;
13. Go to item 3.

It should be noted that procedures "B" and "C" are described in a generalized form - the abstract logical operation is executed in procedure items 9. Replacing the virtual logical operation "OPERATION" on the set of intervals with one of the concrete operations "OR", "AND", "NOT" or "XOR" gives the procedure of performing a concrete logical operation which provides the computation of all edges of the polygon.

5. Procedure for constructing a contour representation

Formally, the result of successive procedures "B" and "C" is not a contour representation, but only the set of all edges of the layer. To obtain contour representation it is necessary to compute all contours [8]. This problem is reduced to the well-known problem of calculating simple cycles in a graph, which can be solved by simply traversing in depth [12] the edge graph built on the found nodes and edges of layer polygons.

The computational complexity of the traversal procedure is linear with the number of edges of the layer polygons and is estimated as $O(N)$. However, the edge graph procedure is computationally more expensive, but does not exceed the estimate as $O(N\ log\ N)$ both in terms of running time and peak memory consumption [10].

6. General algorithm of logical operations on layers

As a result, we get the following scheme for calculating the result of logical operations on the layers:

1. Initial data: contour representation of layer 1 and layer 2;
2. Extract H_1 and H_2 horizontal edges of layer 1 and layer 2
3. Sort the elements of the sets H_1 and H_2 by Y;
4. Perform the procedure «B», get a set of horizontal edges H;
5. Extract V_1 and V_2 vertical edges of layer 1 and layer 2;
6. Sort the elements of the sets V_1 and V_2 by X;
7. Perform the procedure «C», get a set of vertical edges V;
8. Construct an edge graph using the elements from H and V;
9. Find all simple cycles in the depth of an edge graph
10. Save all found cycles to the contour representation of the final layer.

Accumulating information on the complexity of the individual steps of the algorithm, it can be noted that the overall computational complexity of the presented scheme for calculating the results of logical operations on the layers in the contour representation depends only on the number of edges of the layer polygons and is estimated as $O(N\ log\ N)$ in terms of running time and memory consumption [9].

7. Software for DRC

The implementation of algorithms and procedures presented in the work were included in the plug-in library of functions for working with topological description layers, which, in its turn, is a part of the developed system of verification of DRC. This library is implemented in C++, using the C++17 standard. The implementation of the library involves containers from the standard library of C++ templates: vector, set, and algorithms for processing data in the containers.

The general architecture of the system of DRC is shown in Figure 5. In accordance with the architecture, the verification system includes the following components: an interpreter of the Lua language - allows to execute the program of automatic verification; a library of functions for working with layers, which contains functions for performing logical operations; a block of work with GDSII files; a block of work with the database of layers, which contains a block of work with reports.

It should be noted that the input data verification system are the GDSII file of the integrated circuit topology and the verification script. Output data are database of resulting layers and reports of verification algorithms work. Thus, the following possibilities are provided for the verification script: reading layers from the GDSII file; processing layers using functions from the library of functions for working with layers; saving layers to the layer database; saving logs and debugging information to the verification algorithm operation reports repository.

The presented architecture of the developed software provides ample opportunities in terms of expandability of the verification system functionality, which is provided by the large descriptive power of the Lua language within the imperative approach, as well as a wide range of opportunities to increase the functionality of the dynamically connected library of functions.

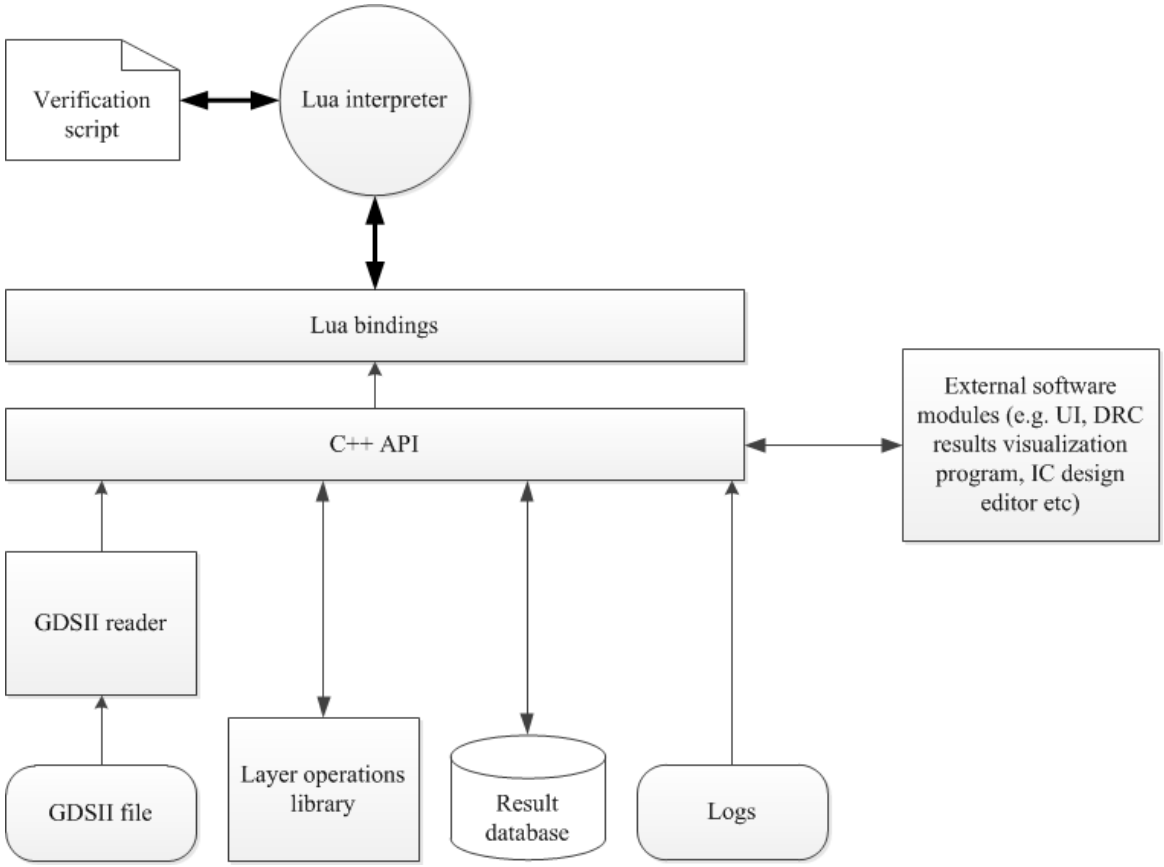

Figure 5: Software architecture for DRC

8. Results of the computational experiment

To test the functions of the library of operations on polygons, we have prepared test data, which is a set of coordinates of the vertices of the contours of polygons for two layers. The data were loaded into the testing application, and two Poly objects were created based on them, over which the operations of union, intersection, subtraction, and exclusive OR were performed. During the experiment, the size of the test data varied from 2000 to 40000 edges making up the contours in one layer.

Macbook Pro with an Intel Core i5 processor, 2.7 GHz, and 8 GB of memory was used for testing.

The results of time measurements for various operations are shown in Table 2, and their visualization is shown in the Figure 6.

Table 2
Time Results

polygon edges number	OR (|) time, sec	AND (&) time, sec	NOT (-) time, sec	XOR (^) time, c
2000	0.2	0.1	0.1	0.2
6000	0.4	0.1	0.3	0.4
10000	1.0	0.3	0.6	1.0
14000	1.3	0.3	0.8	1.3
18000	1.7	0.4	1.1	1.8
22000	2.4	0.5	1.5	2.4
26000	2.8	0.6	1.7	2.8
30000	3.5	0.7	2.1	3.5
34000	4.0	0.8	2.3	4.0
38000	4.5	1.0	2.8	4.6
40000	4.7	1.1	2.9	4.9

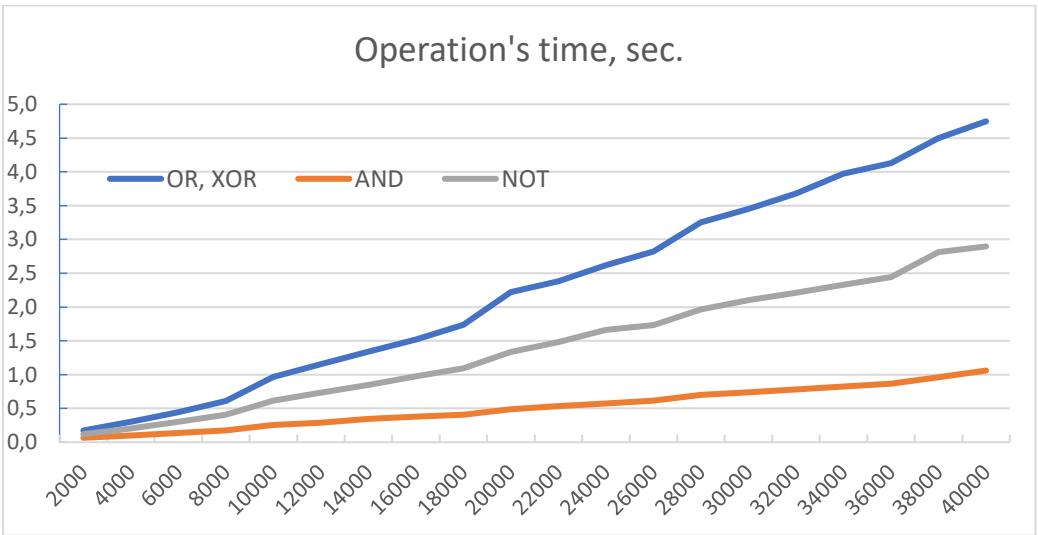

Figure 6: Dependencies of operation time on the number of polygon edges in the layer

The analysis of the obtained dependences allows us to assert that the estimate of the time costs of the operations corresponds to the theoretically obtained estimates. The absolute difference in the time spent on different operations can be explained by the fact that the number of resulting edges at different operations will be different, which leads to different recovery times for the resulting contours. The most time-consuming operations in this sense are OR and XOR, but the intersection operation AND is performed in the least time.

The disadvantage of this implementation is the relatively high absolute time of operations, which requires library code optimization.

9. Conclusion

Thus, within the framework of the project on creation of a domestic CAD for DRC, a library was developed that implements logical operations on the layers that form the topological description of a chip. This library uses a modification of the sweeping line algorithm and properties of orthogonal polygons, which allowed to avoid the quadratic dependence of time on the number of edges in polygons and to achieve quasi-linear dependence both in time and memory consumed by operations. Further work on the library involves the implementation of calculations of various metrics, checks the obtained values for compliance with norms and constraints. Any detected inconsistencies with the rules in the layers will be collected in the verification reports, which serve as the basis for the subsequent correction of errors in the topology.

10. References

[1] A. Shtanyuk, A. Semenov, The problem of integrated circuit topology analysis based on gdsii files, in: Proceedings of the symposium on Engineering and Information Technology, Economics and Management in Industry, Volgograd, 2020. pp. 334-336. (In Russian).

[2] Calibre Design Solutions, 2016. URL: https://eda.sw.siemens.com/en-US/ic/calibre-design.

[3] A. Lohov, The Mentor Graphic company's main caliber, Electronics: Science, Technology, Business 2 (2006) 64-69. (In Russian).

[4] M. I. Shamos, D. Hoey, Geometric intersection problems, in: 17th Annual Symposium on Foundations of Computer Science, 1976, pp. 208-215. doi: 10.1109/SFCS.1976.16.

[5] J. Nievergelt, F. Preparata, Plane Sweep Algorithm for Intersecting Geometric Figures, Communications of the ACM 25 (1982) 739-747. doi: 10.1145/358656.358681.

[6] F. Preparata, M. I. Shamos, Computational geometry: an introduction, Springer-Verlag, 1985.

[7] J. O'Rourke, Computational Geometry in C, Cambridge, University Press, 1998.

[8] J.M. Keil, Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, Handbook of Computational Geometry, pp. 491–518, Elsevier, Amsterdam, 2000.

[9] T. M. Chan, Klee's Measure Problem Made Easy, in: IEEE 54th Annual Symposium on Foundations of Computer Science, 2013, pp. 410-419, doi: 10.1109/FOCS.2013.51.

[10] M. J. Katz, M. H. Overmars, M. Sharir, Efficient hidden surface removal for objects with small union size, Comput. Geom. Theory Appl. 2(4) (1992) 223–234. DOI: 10.1016/0925-7721(92)90024-M

[11] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Section 10.1: Interval Trees, in: Computational Geometry, Second Revised Edition, Springer-Verlag, 2000, pp. 212–217.

[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 2nd. ed., MIT Press and McGraw-Hill, 2001.