FACTA UNIVERSITATIS (NIŠ)
Series: Electronics and Energetics vol. 12, No.1(1999), 87-101

UDC 621.3.049.7

EXTRACTION OF IC CRITICAL AREAS FOR PREDICTING LITHOGRAPHY RELATED YIELD

Zoran Stamenković

Abstract. This paper presents a method for the extraction of integrated circuit critical areas based on a transformation of CIF files from the unrestricted to a restricted format and a local layout extraction approach. The restricted format contains a set of non–overlapping rectangles that abut only along horizontal edges. The extraction of integrated circuit critical areas associated with short and open circuits is carried out by an algorithm that solves this problem time proportional to $n\sqrt{n}$, on average, where n is the total number of the analyzed geometrical objects (rectangles). This algorithm is a typical scanline algorithm with singly–linked lists for storing and sorting the incoming objects. The performance of our method is illustrated on five layout examples by the analysis of CPU time consumed for computing the critical areas applying a software tool system TRACIF/EXACCA/GRAPH.

1. Intoduction

To facilitate failure simulations and integrated circuit yield predictions, the layout information such as minimum spacings and widths, and critical (or sensitive) areas for the conducting layers must be extracted. An extractor to obtain the above layout information automatically is needed. Due to the requirements of the local failure simulation methodology [1], [2], [3], [9], [10], [12], [14], [15] as well as the visual inspection of the critical areas [4],[7], it is convenient to have an extractor, which performance is optimized for local layout extraction.

Manuscript received September 15, 1998.

The author is with Faculty of Electronic Engineering, Beogradska 14, 18000 Niš, Yugoslavia, e-mail: szoran@elfak.ni.ac.yu.

To achieve this goal, we use a simple model for estimating integrated circuit critical areas, and internal data structures for storing the geometrical objects (rectangles) of a circuit layout and these critical areas. In this paper a local critical area extraction approach is described. Moreover, the extraction algorithm and the implementation details for both the front–end and back–end of the extraction system are presented. Finally, possible applications and extensions of this system are discussed.

2. Local extraction approach

2.1 A model for estimating IC critical areas

Two most significant types of primitive faults in integrated circuits related to very small critical dimensions of integrated circuit layout patterns are short and open circuits caused by lithographic defects. It is obvious that defects are iregular rough edged splotches that can be modeled using circular objects. The critical areas for short and open circuits can be defined as areas in which the center of a circular defect must fall to cause one of these faults (Fig. 1). These areas consist of a rectangular part and four equal circular parts, and are functions of the defect diameter x. Expressions for the critical areas $A_s(x)$ and $A_o(x)$ have been derived by Stapper [18] and Ferris–Prabhu [8] for a number of long conducting lines. Koren [13] and Corsi [5] make effort to model the critical area for realistic integrated circuit patterns, whose lengths are not much longer than their widths and spacings. However, these attempts to take into account the edge efects of conducting lines by approximating the circular parts of the critical areas $A_{s\bigcirc}(x)$ and $A_{o\bigcirc}(x)$ with a triangle have not enabled modeling the critical area quite correct. Namely, the approximation of the circular part with a triangle causes the neglection of the segment of a circle of the diameter x, whose area can be over 30% of the total area of the circular part.

Therefore, we propose a new expression for definition of the circular part of critical area for short circuit between two geometrical objects at the distance s (see Fig. 1(a))

$$A_{s\bigcirc}(x) = A_{s\triangle} + A_{s\oslash} = \frac{x-s}{8}\sqrt{x^2-s^2} + \frac{x^2}{4}\left(\arcsin\sqrt{\frac{x-s}{2x}} - \sqrt{\frac{x-s}{2x}}\cos\arcsin\sqrt{\frac{x-s}{2x}}\right), \quad (1)$$

and a new expression for definition of the circular part of critical area for

opening a geometrical object of the width w (see Fig. 1(b))

$$A_{o\bigcirc}(x) = A_{o\triangle} + A_{o\oslash} = \frac{x-w}{8}\sqrt{x^2-w^2} + \frac{x^2}{4}\left(\arcsin\sqrt{\frac{x-w}{2x}} - \sqrt{\frac{x-w}{2x}}\cos\arcsin\sqrt{\frac{x-w}{2x}}\right), \tag{2}$$

where $A_\triangle$ is the area of triangle ACF, $A_\oslash$ is the area of circle segment $A\frown C$, and $x > s, w$.

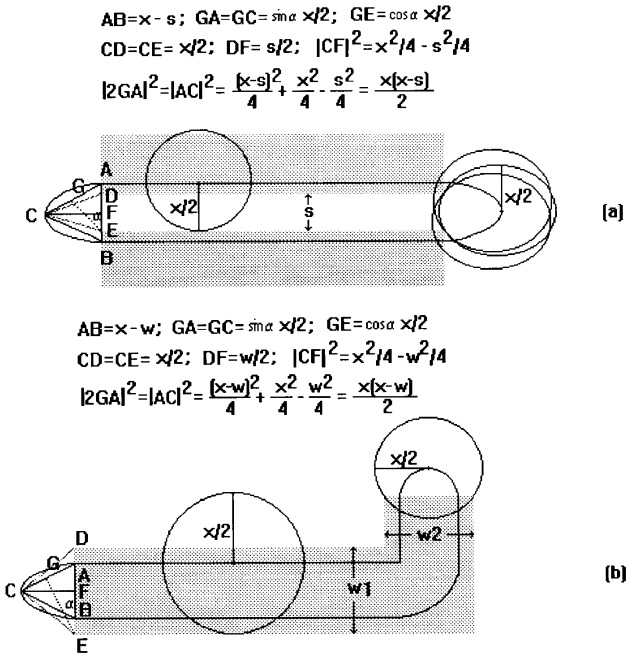

Fig. 1. The critical areas for short (a) and open (b) circuits.

The estimation of the critical area associated with lithographic defects requires averaging with respect to the defect size distribution as follows [18]

$$\bar{A} = \int_0^\infty A(x)h(x)dx, \tag{3}$$

where $A(x)$ (A_s or A_o) is the critical area associated with defects of a given size and $h(x)$ is the defect size distribution. It should be emphasized here

that a significant disagreement between the defect size distribution approximated by the conventional $1/x^3$ function and the measured data has been observed (Fig. 2).

Therefore, we suggest the Gamma distribution function, which is normalized itself, to describe the defect size distribution [16]. The most important region of the defect size distribution in view of the critical area calculation is so called *tail* region. Because of that the insert of Fig. 2 shows a suitable view of the tail region of the defect size distributions approximated by the $1/x^3$ function and the Gamma distribution function. As can be seen from Fig. 2, compared to the $1/x^3$ function, the Gamma distribution function provides necessary quick decrease of the defect size distribution for the higher defect sizes ($> 8.5\mu m$) and much better agreement of the defect size distribution with the measured data for the lower defect sizes ($< 8.5\mu m$).

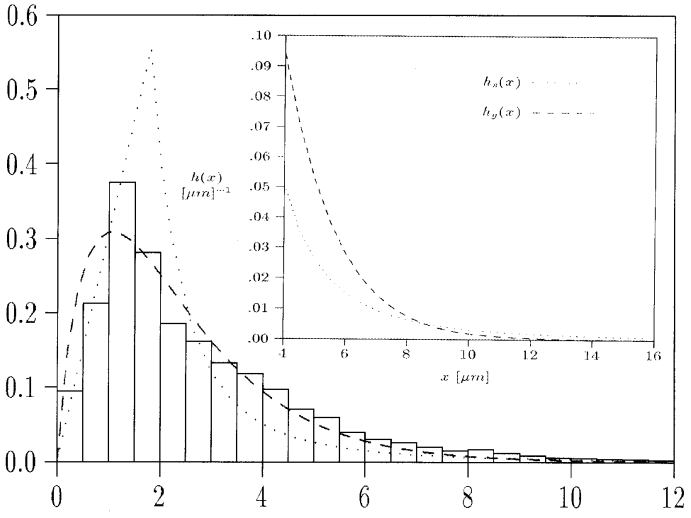

Fig. 2. Empirical distribution (histohram);
Gamma distribution $(- - -)$*;* $1/x^3$ *distribution* $(\cdots)$*.*

2.2 Data structures

The simplest way to extract the critical area for shortening and opening geometrical objects is the comparison of a geometrical object to all the other geometrical objects. This is computationally prohibitive in the case of modern integrated circuits, which can contain tens of millions of transistors due to its $O(n^2)$ performance, where n is the total number of objects. There-

fore, algorithms that enable an efficient processing of geometrical objects and minimization of the number of comparisons between object pairs must be used. These algorithms are more complex than $O(n)$ and their complexity determines the CPU time and memory consumption.

We use a typical scanline algorithm with a list for storing the incoming objects whose top edges coincide with the scanline. Then every object in this list is sorted and inserted into another list called an *active list* and similar to that in [17]. In the meantime, layout extractions are done by comparing the object being inserted to other objects in the active list. An object then exits the active list when the scanline is at or below its bottom edge. The choice of a data structure for efficient geometrical object representation plays an important role. Since local extraction methodology is chosen, a good candidate for the data structure requires a fast region query operation and a reasonable memory consumption.

Three kinds of data structures are needed for the critical area extraction. The first one is used for efficient object representation in the active list. To minimize the number of comparisons between object pairs, a suitable structure should be developed so that extraction can be performed as locally as possible. A singly–linked list is chosen for the active list not only for its simplicity, but also for its speed and memory efficiency. The chosen singly–linked list and corresponding data structure are described in Fig. 3(a). The proposed data structure contains fields $X1, X2, Y1$ and $Y2$ which represent the coordinates of the left, right, bottom and top edges of a rectangle, respectively, and two additional fields called REC and MK used to indicate the rectangle number and the rectangles of the same pattern. The comparisons between active rectangles stored in the active list can be carried out as locally as possible by examining the sorted coordinates of rectangles. The second data structure is used for a list of coordinates of the critical area for shortening two rectangles from different patterns. The proposed data structure is shown in Fig. 3(b) and contains fields $x1, x2, y1$ and $y2$, which represent the coordinates of the left, right, bottom and top edges of the critical area, respectively, a field $\bar{A}_s$, which represents the value of critical area itself, and two additional fields called $REC1$ and $REC2$ used to indicate the rectangle numbers. The third data structure is used for a list of coordinates of the critical area for opening a rectangle. The proposed data structure is shown in Fig. 3(c) and contains fields $x1, x2, y1$ and $y2$, which represent the coordinates of the left, right, bottom and top edges of the critical area, respectively, a field $\bar{A}_o$, which represents the value of critical area, and an additional field called REC used to indicate the rectangle number.

3. Algorithm for critical area extraction

The main tasks of described approach are to find out *all objects narrower than the largest defect with the diameter* x_{max}, *all pairs of objects with a spacing between them shorter than the largest defect diameter* x_{max}, to determine canonical coordinates of the critical areas $(x1, y1)$ and $(x2, y2)$ for the largest defect diameter x_{max}, and to compute the critical areas by making use of the expression (3).

Therefore, we have developed the algorithm for local critical area extraction based upon the scanline method for scanning the sorted geometrical objects and the singly–linked list for representation of the active list of geometrical objects. The main steps of the algorithm are as follows:

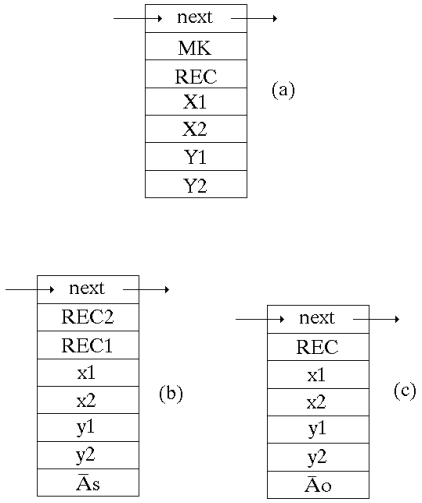

Fig. 3. The data structure for a geometrical object representation in the active list (a), the data structure for representation of the critical area for shortening geometrical objects (b), and the data structure for representation of the critical area for opening a geometrical object (c).

ALGORITHM

- Input: a singly–linked list of rectilinearly oriented rectangles sorted according to the top edges from top to bottom from the same integrated circuit mask layer
- Output: critical areas for short and open circuits between rectangles from

the same integrated circuit mask layer

1. Set the scanline to the top of the first rectangle from an input list;
2. WHILE (the scanline $\geq$ the top of the last rectangle from an input list)

1. Update the active list **SOR**;
2. Fetch rectangles from an input list whose the top coincides with the scanline and store them in a singly–linked list called **TR**;

Update the scanline;
FOR each new rectangle in **TR**

1. **Seek/Left()** sorts the new rectangle and inserts it into **SOR**, computes $A_{o\bigcirc}$ and $A_{s\bigcirc}$ for the new rectangle and rectangles from **SOR** left to it, and computes and stores the critical areas for short and open circuits in two singly–linked lists;

Seek/Right() computes $A_{o\bigcirc}$ and $A_{s\bigcirc}$ for the last inserted rectangle into **SOR** and rectangles from **SOR** right to it, and computes and stores the critical areas for short and open circuits in two singly–linked lists;

3. Write the critical areas into output files.

The scanning process starts with setting the scanline to the top edge of the first rectangle from an input list. The second step is a loop for updating the active list and moving the scanline. To update rectangles in **SOR**, Substep 2.1 of the above algorithm performs comparison between the current scanline and the bottom edges of rectangles in **SOR**. If the bottom edge of a rectangle is above the current scanline for a *threshold* value (in this case, the largest defect diameter x_{max}) or more, a rectangle will be deleted from **SOR**. This guarantees that the critical areas for short circuit between any two rectangles in the y direction can be detected. Substep 2.2 makes a singly–linked list **TR** contained rectangles with the same y coordinates of the top edges. This step enables to sort rectangles according to the x coordinate of the left edge. The y coordinate of the next scanline (Substep 2.3) is equal to the top edge of the next rectangle in an input list. Substep 2.4 sorts and inserts each new rectangle from **TR** into the active list **SOR**, and computes and stores the critical areas in output lists. The last step of the algorithm writes the content of output lists, i.e. coordinates of the critical areas for short and open circuits $(x1, y1)$ and $(x2, y2)$ for the largest defect diameter x_{max}, as well as values of the critical areas $\bar{A}_s$ and $\bar{A}_o$ in two output files.

Procedure **Seek/Left()** takes the new rectangle from **TR** and the active list **SOR** as inputs and reports the critical areas as output. Rectangles are sorted by the comparison of their left edge coordinates $X1s$. The sorted

rectangles are stored in the active list **SOR**. The main steps of this procedure are as follows:

Seek/Left()

FOR each new rectangle in **SOR**
IF $(X1(TR) \leq X1(SOR))$
Insert the new rectangle from **TR** into **SOR** just before the current rectangle SOR;
RETURN;
ELSE IF $(X1(TR) < X2(SOR) + x_{max})$
IF $(X1(TR) \leq X2(SOR)$ && $Y1(SOR) \leq Y2(TR))$
CompL/CrArO;
MK(SOR) = REC(TR);
MK(TR) = REC(SOR);
ELSE IF $(X1(TR) \leq X2(SOR)\&\&MK(SOR) \neq MK(TR))$
$s = Y1(SOR) - Y2(TR)$;
CompL/CrArS;
ELSE IF $(Y1(SOR) \leq Y2(TR)\&\&MK(SOR) \neq MK(TR))$
$s = X1(TR) - X2(SOR)$;
CompL/CrArS;
ELSE IF $((X1(TR) - X2(SOR))^2 + (Y1(SOR) - Y2(TR))^2 \geq x_{max}^2)$
CONTINUE;
ELSE IF $(MK(SOR) \neq MK(TR))$
CompL/CrArS;
Insert the new rectangle from **TR** into the tail of **SOR**.

CompL/CrArO computes coordinates of the critical area for open circuit $(x1, y1)$ and $(x2, y2)$ for the largest defect diameter x_{max}, as well as a value of the critical area $\bar{A}_o$ by the expression (3) for the new rectangle from **TR** and the current rectangle SOR, and insert the new node into an output list called **CO**. *CompL/CrArS* computes coordinates of the critical area for short circuit $(x1, y1)$ and $(x2, y2)$ for the largest defect diameter x_{max}, as well as a value of the critical area $\bar{A}_s$ by the expression (3) between the new rectangle from **TR** and the current rectangle SOR, and insert the new node into an output list called **CS**.

Procedure **Seek/Right()** takes the last inserted rectangle into **SOR** and **SOR** itself as input and reports the critical areas as output. In a loop of this procedure, the place of the last inserted rectangle *SOR is checked first by the comparison of its right edge coordinate $X2$ with the left edge coordinate $X1$ of the current rectangle SOR. It enables to end this loop earlier. The main steps of this procedure are as follows:

Seek/Right()

FOR each new rectangle in **SOR** starting at the last inserted rectangle *SOR
IF $(X2(*SOR) + x_{max} < X1(SOR))$
RETURN;
ELSE IF $(X1(SOR) \leq X2(*SOR) \&\& Y1(SOR) \leq Y2(*SOR) \&\& SOR != *SOR)$
CompR/CrArO;
MK(SOR) = REC(*SOR);
MK(*SOR) = REC(SOR);
ELSE IF $(X1(SOR) \leq X2(*SOR)$ && $MK(SOR) \neq MK(*SOR))$
$s = Y1(SOR) - Y2(*SOR)$;
CompR/CrArS;
ELSE IF $(Y1(SOR) \leq Y2(*SOR)$ && $MK(SOR) \neq MK(*SOR))$
$s = X1(SOR) - X2(*SOR)$;
CompR/CrArS;
ELSE IF $((X1(SOR) - X2(*SOR))^2 + (Y1(SOR) - Y2(*SOR))^2 \geq x_{max}^2)$
CONTINUE;
ELSE IF $(MK(SOR) \neq MK(*SOR))$
CompR/CrArS;
ELSE
CONTINUE;

CompR/CrArO computes coordinates of the critical area for open circuit $(x1, y1)$ and $(x2, y2)$ for the largest defect diameter x_{max}, as well as a value of the critical area $\bar{A}_o$ by the expression (3) for the last inserted rectangle *SOR and the current rectangle SOR, and insert the new node into an output list called **CO**. *CompR/CrArS* computes coordinates of the critical area for short circuit $(x1, y1)$ and $(x2, y2)$ for the largest defect diameter x_{max}, as well as a value of the critical area $\bar{A}_s$ by the expression (3) between the last inserted rectangle *SOR and the current rectangle SOR, and insert the new node into an output list called **CS**.

Note that geometrical objects, i.e. rectangles from the same integrated circuit mask layer have to be stored in the active list **SOR**. A simple example, which illustrates the proposed algorithm is described in Fig. 4. The figure shows rectangles in the active list with scanlines shown in sequence. When the scanline reaches the position **S1** the newest rectangle in the active list **SOR** is the rectangle 6. In the meantime, the critical area for opening this rectangle and the critical areas for shortening it with the rectangles 3 and 4 are computed. As the scanline moves down, its next stopping position is **S2**. Now the newest rectangle in the active list is the rectangle 7. In the same time, the rectangle 4 exits the active list **SOR** because *the spacing between its bottom edge and the current scanline is greater than a threshold value* x_{max}. By making use of the described algorithm, the critical areas related

to lithographic defects for any integrated circuit conducting layer can be extracted.

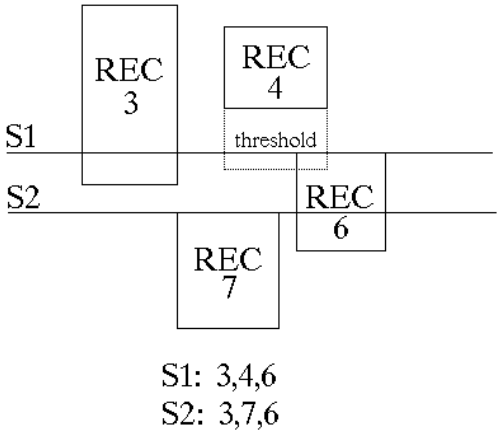

Fig. 4. Scanlines with rectangles in the active list **SOR**.

4. Implementation and performance analysis

The layout extraction starts from the layout description in CIF format and ends by reporting the critical areas for short and open circuits. In our case, this procedure is done through a software system, which consists of three tools. The above algorithm is only dedicated to the back–end of the entire system and is implemented in a program called EXACCA (EXtrActor of Chip Critical Area). The structure of this system is shown in Fig. 5.

The front–end of system is a technology independent processor for transforming integrated circuit layout description from the unrestricted to a restricted format TRACIF (TRAnsformer of CIF) [11]. The unrestricted format can contain overlapping rectangles, as well as rectangles making bigger rectangles from the same integrated circuit mask layer. However, an internal restricted geometric representation should contain a set of non–overlapping rectangles that abut only along horizontal edges. There are two important properties of the restricted format:

Coverage - Each point in the $x - y$ plane is contained in exactly one rectangle. In general, a plane may contain many different types of rectangles.

Strip - Patterns of the same integrated circuit mask layer are represented with horizontal rectangles (strips) that are as wide as possible, then as tall as possible. The strip structure provides a canonical form

for the database and prevents it from fracturing into a large number of small rectangles.

Fig. 5. The critical area extraction system.

TRACIF takes a CIF file as an input and generates files containing geometrical objects (rectangles) defined by the canonical coordinates of each integrated circuit cell and mask layer as outputs. Thus the outputs of TRACIF are lists of sorted rectangles according to the top edges from top to bottom. TRACIF can handle Manhattan shaped objects and consists of about 800 lines of C code. Therefore, TRACIF is capable to perform the layout description transformation hierarchically. Namely, TRACIF transforms a CIF file to the restricted format in a hierarchical way and makes different files for different cells and layers. This feature is desirable since most of the modern integrated circuit designs exploit the technique of design hierarchy. Within this design methodology, the layout extraction is only required once for each layout cell.

EXACCA takes the sorted rectangles and starts the critical area extraction by using the algorithm proposed in Section 3. EXACCA can handle Manhattan type objects and consists of about 2000 lines of C code. The outputs of EXACCA are lists of the critical areas for short and open circuits. The visual presentation of the critical areas has been performed by the software tool GRAPH. A pictorial example of the layout and snapshots of the corresponding critical areas is shown in Fig. 6. Precision of the visual presentation of the critical areas is limited by the error made in approximation of the circular parts by rectangular subareas and equal to precision of the method presented in [7].

To analyze the performance of the algorithm, an idealized model is used. If there are n uniformly distributed rectangles in a region of interest, there will be around $\sqrt{n}$ rectangles, on average, in each scanline. Based upon this model, the time complexity of the algorithm is analyzed. Step 1 in the algorithm is trivial and takes a constant time. Step 2 is a loop with, on average, $\sqrt{n}$ elements under which there are four substeps. Substeps 1 and 2 take $O(\sqrt{n})$ expected time due to their $\sqrt{n}$ length of elements in **SOR** and **TR**. Substep 3 takes a constant time. Substep 4 has $\sqrt{n}$ elements in a loop under which subsubsteps 1 and 2 take $O(\sqrt{n})$. Hence, Substep 4 takes $O(n)$ time. As a result, Step 2 takes $O(n\sqrt{n})$ time. Note that the critical areas

Fig. 6. The layout (a) and the critical areas for short (b) and open (c) circuits for the first metal layer of operational amplifier

$\bar{A}_s$ and $\bar{A}_o$ are calculated using Simpson's method for numerical integration with the x–resolution $0.1\mu m$. Finally, Step 3 takes a constant time. From the above idealized analysis, the complexity of this algorithm is $CO(n\sqrt{n})$, where C is a constant. The approaches used in [6], [7], [10] promise $O(n \log n)$ performance, even though authors note that the actual consumption of CPU time is a very intensive.

Since a VLSI circuit can contain tens of millions of transistors, the limitation of memory resources places an important role on extraction efficiency. To avoid running out of memory, special coding techniques have to be employed. These techniques decrease the extraction efficiency, particularly for very big circuits. In general this memory limitation problem affects the algorithms regardless of which data structure is used for the node representation of the active list. However, a singly–linked list suffers the least due to its memory efficiency. Thus a list structure is preferred as far as memory space is concerned. The memory consumption of EXACCA is proportional to $\sqrt{n}$.

Here the simulated results of five examples which were designed using double metal CMOS process will be presented. The first one was designed by the students and the last four by Tanner Research Centre. The number of rectangles, as well as the CPU time of EXACCA on Silicon Graphics Indy workstation for these five integrated circuit layouts called *chip, counter4, counter6, counter8* and *counter10* are shown in Table 1. The extraction speed is illustrated by the analysis of CPU times needed for the computation of critical areas for short and open circuits for five values of the largest defect diameter x_{max}. The extraction results show that a CPU time increases as the largest defect diameter increases. Namely, the increase of the largest defect diameter means a greater threshold for updating the active list and, consequently, a greater number of rectangles in the active list **SOR**. The increase of the number of comparisons between rectangle pairs causes corresponding increase of the extraction time of critical areas. As can be seen from Table 1, one of the most important advantages of the proposed extraction algorithm and corresponding data structures is the ability to process large layouts in a relatively short CPU time.

Table 1. Extraction time on Silicon Graphics Indy workstation

Integrated circuit	Number of rectangles	CPU time (s) for x_{max}				
		$10\mu m$	$12\mu m$	$14\mu m$	$16\mu m$	$18\mu m$
chip	4125	65.1	83.7	89.4	97.8	112.0
counter4	11637	342.4	379.5	406.7	459.9	503.1
counter6	19503	600.2	631.4	685.5	792.6	885.9
counter8	24677	897.6	954.2	1083.6	1217.8	1399.8
counter10	30198	1116.0	1338.1	1542.2	1689.5	1880.3

5. Applications

EXACCA ensures the microscopic layout information needed for more detailed analysis. Thus the output of EXACCA may be used for any integrated circuit yield simulation system and design rule checking system. Also, our software system can be useful for the macroscopic yield models, which require to know the chip critical area. Caution should be taken in this case as the total chip critical area must be computed by finding the union of (not by adding) the critical areas.

Regardless of the fact that we have only focused on the extraction of critical areas, EXACCA can also be easily modified for the extraction of parasitic effects. While the critical areas are required for the simulation of functional failures, the extraction of parasitic effects can be used for the simulation of performance failures.

REFERENCES

1. G.A. Allan, A.J. Walton and R.J. Holwill: *An yield improvement technique for IC layout using local design rules.* IEEE Trans. Computer–Aided Design of ICAS, vol.11, pp. 1355–1362, 1992.
2. G.A. Allan and A.J. Walton: *Sampling based yield prediction for ULSI.* Proc. SPIE Symposium on Microelectronic Manufacture, Austin, 1996, pp.198–209.
3. I. Chen and A. Strojwas: *Realistic yield simulation for VLSIC structural failures.* IEEE Trans. Computer-Aided Design of ICAS, vol.6, pp.965–980, 1987.
4. V.K.R. Chiluvuri and I. Koren: *Layout–synthesis techniques for yield enhancement.* IEEE Trans. Semiconductor Manufacturing, vol.8, pp.178–187, 1995.
5. F. Corsi et al.: *Critical areas for finite length conductors.* Microelectronics and Reliability, vol.32, pp.1539–1544, 1992.
6. A. Dalal, P. Franzon and M. Lorenzetti: *A layout–driven yield predictor and fault generator for VLSI.* IEEE Trans. Semiconductor Manufacturing, vol.6, pp.77–82, 1993.
7. J.P. deGyvez and C. Di: *IC defect sensitivity for footprint-type spot defects.* IEEE Trans. Computer–Aided Design of ICAS, vol.11, pp.638–658, 1992.
8. A.V. Ferris–Prabhu: *Modeling the critical area in yield forecasts.* IEEE J. Solid-State Circuits, vol.20, pp.874–877, 1985.
9. D.D Gaitonde and D.M.H. Walker: *Hierarchical mapping of spot defects to catastrophic faults – design and applications.* IEEE Trans. Semiconductor Manufacturing, vol.8, pp.167–177, 1995.
10. F.M. Goncalves et al.: *Integrated approach for circuit and fault extraction of VLSI circuits.* Proc. IEEE International Symposium on Defect and Fault Tolerance, 1996.
11. D. Janković, D. Milenović and Z. Stamenković: *Transforming IC Layout Description from The Unrestricted to A Restricted Format.* Proc. 21st International Conference on Microelectronics, Niš, 1997, pp.733–735.

12. J. Khare and W. Maly: *Rapid failure analysis using contamination–defect–fault (CDF) simulation.* IEEE Trans. Semiconductor Manufacturing, vol.9, pp.518–526, 1996.

13. I. Koren: *The effect of scaling on the yield of VLSI circuits.* Yield Modeling and Defect Tolerance in VLSI, edited by W. Moore, W. Maly and A. Strojwas, Bristol, 1988, pp.91–99.

14. J.H.N. Mattick, R.W. Kelsall and R.E. Miles: *Improved critical area prediction by application of pattern recognition techniques.* Microelectronics and Reliability, vol.36, pp.1815–1818, 1996.

15. P.K. Nag and W. Maly: *Hierarchical extraction of critical area for shorts in very large scale ICs.* Proc. IEEE Workshop on Defect and Fault Tolerance in VLSI Systems, Lafayette, 1995, pp.19–27.

16. Z. Stamenković and N. Stojadinović: *New defect size distribution function for estimation of chip critical area in integrated circuit yield models.* Electronics Letters, vol.28, pp.528–530, 1992.

17. Z. Stamenković: *Algorithm for extracting overlap areas between integrated circuit mask layers.* Proc. 8th IASTED International Conference on Reliability, Quality Control and Risk Assessment, Washington,DC, 1992, pp.156–159.

18. C.H. Stapper: *Modeling of defects in integrated circuit photolithographic patterns.* IBM Journal Research and Development, vol.28, pp.461–475, 1984.