IMPLEMENTATION OF AN EFFICIENT ALGORITHM FOR VLSI DESIGN RULE CHECKING ON A 2-D MESH OF TRANSPUTERS

Daniele D. Caviglia, Maurizio Paganini, Marco Chirico, Francesco Curatelli, Marco Barzaghi and Giacomo M. Bisio

DIBE - University of Genoa - via Opera Pia 11/a - 16145 Genoa - ITALY

ABSTRACT

This paper describes a Design Rule Checking (DRC) program for VLSI circuit layouts with Manhattan geometries. We present the layout model, the way we implemented the check through few and simple primitives for describing parametric rules, a single main function for the geometrical test, how we succeeded in limiting the checking time through an algorithm characterized by a linear computational complexity and how we mapped it on a mesh of transputers.

I - INTRODUCTION

The layout generation is one of the most error prone and time consuming tasks occurring in the design of VLSI circuits. Thus it must be supported by automatic verification of its correctness according to the design rules set of the technology. Earlier approaches tried to check the layout pixel by pixel through the definition of a suitable pixel window; this window is made up by a number of pixels around the one to be checked. Obviously, it is necessary to define an area which is sufficient to investigate the correctness of the actual pixel (central pixel) [1,2]. In these approaches, the Design Rules are associated to Error Patterns, i.e. particular configurations for pixel windows [3].

More recent approaches are intended to take advantage of a hierarchical definition of the layout to be checked, in which the layout is partitioned into a number of cells; each cell can then be replicated and translated or rotated in order to be correctly displaced in the layout [4,5,9]. This kind of layout can be efficiently checked considering each cell once. However, these approaches have to face the big problem of manipulating cells overlaps, which is particularly difficult when connections are allowed to pass through or over the cells.

In this paper, a solution is presented which partitions a flattened layout into rectangular areas which are checked separately and independently. The program has been effective in checking very complex layouts. Moreover the geometrical tests are simplified by using a single main check function. This approach leads to an easy implementation in coarse-grain parallel computers. The layout model is described in Section II. The rule checking task and the algorithm complexity are outlined in Section III and IV. The algorithm itself and its mapping on a mesh of transputers are reported in Section V. Results and conclusions are in Sections VI and VII.

II - THE LAYOUT MODEL

A layout is defined as a collection of potentially overlapping geometrical shapes laying on different layers. It is worth noting that these layers have not any a priori significance, in the sense that no predefined information about the technology is required. In this way, every type of technology can be handled by the program provided that an external association of real masks layers to such symbols is done.

The model we have adopted is able to manage layouts with standard Manhattan geometries, i.e. layouts composed only by rectilinear poligons. The program accepts as input a description of the layout in CIF format and transforms it into a set of layout planes, each one representing the collection of all geometrical shapes belonging to a specific symbolic layer. Moreover, within each plane, the program converts all the geometrical shapes constituting the layout into rectangles. This is done in order to simplify the check process.

III - THE RULE CHECKING TASK

The geometrical shapes in the layout must satisfy a series of topological constraints (Design Rules). Three types of design rules are considered:

1) width rules, which indicate the minimum or maximum extension of a shape belonging to a layer
2) distance rules, which indicate the minimum distance between two shapes of the same layer or of different layers
3) overlap rules, which indicate the minimum extension of the shapes of a layer inside or outside the shape of another layer

In order to handle properly all the different configurations which characterize a typical technology, it is necessary to define complex layers. We eventually defined seven operations on the rectangles of a layer or couples of layers:

logical:
Complement: $P_{\text{not}} = \overline{P}$
Intersection: $P_{\text{int}} = P_1 \& P_2$
Union: $P_{\text{un}} = P_1 \lor P_2$
Containment: $P_{\text{cont}} = P_1 \# P_2$

geometrical:
Enlargement: $P_{\text{en}} = P + w$
Minimization: $P_{\text{min}} = P < w$
Maximization: $P_{\text{max}} = P > w$

where $w$ is a width (in our case in hundredth of micron).

The check of the design rules has been approached in a simple way verifying the distance between two rectangles. If we have to check a width or an overlap rule, we generate the complementary plane of the layer under test and use the correspondence existing between their rectangles.
Each overlap rule can be transformed into a directional width rule and checking the correct width of a rectangle belonging to a layer is equivalent to check the distance among the rectangles of the corresponding complementary plane surrounding it (see Fig. 1 and 2).

**IV. ALGORITHM COMPLEXITY**

The algorithm we have applied to the geometrical entities derived from the operations described so far, features a quadratic dependence on the number of rectangles which form the layout, since each rectangle can be related to all the others. This number is nowadays quite relevant (about $10^5$) so that the resulting checking times are impractical. Due to the fact that the geometrical shapes own a good level of locality, i.e. their dimensions are at least one order of magnitude smaller than those of the whole layout, and that at the same time the design rules refer to overlapping or, however, neighbouring shapes, we have partitioned the layout in rectangular areas (referred to as sectors).

Each sector is independent from all the others thanks to the definition of a boundary strip whose width is a function of the design rules (see Fig. 3). This strip is owned by both sectors facing to it, and the rectangles in it are checked twice (once for each sector) or four times (the rectangles positioned on the boundary strips crosses).

Although the number of rectangles checked with this solution is $N_t = \gamma N$, with $1.5 < \gamma < 3$, which is greater than the number of rectangles in the layout ($N$), the new complexity of the algorithm is linear with $N$; this is due both to the way the layout is partitioned (the average number of rectangles checked by each sector is constant) and to the good locality of the distribution of the rectangles in the sectors.

In fact being $K$ the total number of sectors, the average number of rectangles per sector is $n = \frac{N_t}{K}$ and consequently the complexity for each sector is $O\left(\frac{N_t^2}{K}\right)$.

The resulting checking time ($T$) for the whole layout is then

$$T = O\left(K \times \left(\frac{N_t}{K}\right)^2\right) = O\left(\frac{N_t^2}{K}\right) = O\left(\frac{N_t^2}{N_t} \times n\right) = O\left(n\right).$$

It has been shown through experiments that an optimal number $n (n_{opt})$ exists for a given technology and it is almost independent from the layout and is related to the design rules (Fig. 4).
V - ALGORITHM PARTITIONING AND TASK ALLOCATION ON TRANSPUTERS

The DRC program execution can be divided into three different phases:

a) Layout Flattening and Decomposition into Sectors:

This phase is performed once and generates a data structure which represents a collection of sectors from a layout in CIF format. Its complexity is O(N).

b) Sectors Analysis:

It consists of the analysis exposed in Section IV performed on each sector by the same program code. This fact suggests the implementation of a data decomposition strategy. By this way, the analysis can be performed in parallel on a mesh of M transputers implementing a master-slave mechanism with one master processor and M-1 slave processors.

c) Error Report Handling:

This phase is performed during the check and consists in writing into an output file the results of the analysis. The time for the execution of this phase is usually negligible.

Owing to what we have outlined for phase b), we have mapped the algorithm on a set of transputers organized following the master-slaves scheme. The particular organization we have adopted consists in a single master processor from which descends a binary tree whose leaves are the slave processors (see Fig. 5).

The master processor executes the partitioning of the layout and submits the data sets of each sector to the available slave processors for the execution of the verification. If the number of sectors is less than or equal to the number of slave processors, no more data submission is required (see Fig. 6). If the number of sectors is greater than the number of slave processors, the master processor waits for a slave request and submits data until all the sectors have been verified.

After each sector verification, the slave processor sends to the master processor the results (detected errors) if any, and waits for the new data. At the end of the verification process, the master generates the complete error report.

VI - RESULTS

Our attention has focused on two main items: comparing the performance of the DRC with the equivalent SUN version and calculating the speed-up and efficiency of the transputer network. As far as the first item is concerned, we have noticed that the transputers network suffers a great deal the weight of data transfer between the master and the slaves. This can be clearly seen in Figure 7. Obviously the greater the layout is, the more such a limitation affects the transputers network efficiency.

The results concerning the speed-up item are quite interesting. Figure 8 shows what has been obtained; in this case too, the difference between the ideal curve and the real one is determined by communications overhead.

Table 1 contains the summary of speed-up and efficiency values for transputer networks of different sizes; it should allow a full analysis of the performance of the solution we have adopted.
VI. CONCLUSIONS AND FUTURE WORK

A DRC program has been developed and tested. It is based on an algorithm characterized by a linear computational complexity. This algorithm implements a layout partitioning into sectors which can be processed in parallel. Future work will concern the study of different network topologies which can limit the intrinsic communications overhead.

REFERENCES.


