SPECIAL PURPOSE ARCHITECTURE FOR ACCELERATING BITMAP DRC

N.B.Bhat

Elec. Comm. Engg.
Indian Institute of Science
Bangalore 560 012
INDIA

S.K.Nandy

Laboratory for CAD
Indian Institute of Science
Bangalore 560 012
INDIA

Abstract *In this paper we propose algorithms for performing DRC on a bitmapped layout and develop special purpose architecture for its implementation. We use window scan method, with flexible window size, reduce redundant pixel processing in a window unlike that in [5].*

1. Introduction

Design Rule Checking (DRC) of layout geometries is an important phase in LSI/VLSI designs. Blank [3,4] proposed a Virtual Bit Map Processor(VBMP), to perform DRC by means of Shrink/Expand instructions. The number of instructions required to perform DRC is linearly proportional to the length of the rule being checked. Seiler [5] describes a window processor architecture implementation for a rasterized layout. The window processing effectively moves a 3×3 or 4×4 window across the entire layout, and checks if the bitmap pattern within the window is valid. In this scheme, each pixel of the bitmap is processed n^2 times for a window of size $n \times n$. The algorithm also takes care of non-orthogonal geometries.

The algorithm proposed in this paper is based on a window scan and aims at reducing processing of a bitmap pixel. We restrict the layout to rectangular geometries. We keep window size (referred as the partition) flexible, and achieve faster processing with larger windows. Size of the window is limited by the number of basic DRC chips (described in the text of the paper) that are connected together. The time required to process a window is independent of the window size. All design checks are carried out one rule at a time and to facilitate valid error reporting at the window boundaries, adjacent windows must overlap. The overlap between two adjacent windows is at least the length of design rule being checked. A pixel lying in the region where two or more partitions overlap is processed maximum 4 times, and other pixels are processed only once.

To implement the DRC, within a partition a highly parallel algorithm is used which is realized in hardware. The algorithm performs width and space checks simultaneously with a very small instruction set. Specifically, the design rule check on a layer is performed using only a single instruction.

2. Definition

We now give a few definitions of terms used with regard to a bitmap representation of a single layer in the layout (Fig. 2.1). **Region within a layout polygon** is the region which lies to the left of the polygon boundary in the anti-clockwise direction. **Pixel** is an elementary square area with its length and width $= \lambda$, where λ is the unit in which the design rules are expressed. A pixel is called **Black pixel** if it forms a part of a region within any layout polygon; otherwise it is called **White pixel. Edge pixel** is a black pixel which forms the polygon boundary and is identified by the edge of the polygon boundary. We have left, right, top and bottom edge pixels corresponding to the left, right, top or bottom edge of the pixel belonging to the polygon boundary, respectively. Identifying edge pixels in this manner, however, does not imply that it is the *only* edge of the pixel belonging to the boundary. A **corner pixel** is a black pixel at the corner of a layout polygon. A **corner vertex** is the vertex of a corner pixel that forms a layout polygon corner. A corner pixel can have more than one corner vertices. A **concave corner pixel** is the corner pixel whose at least two edges form a part of a layout polygon boundary. All concave corner pixels are edge pixels; the converse is not always true. A **convex corner pixel** is the corner pixel whose no edge forms a part of a polygon boundary. **Corner diagonal** is the diagonol drawn at a corner vertex of a corner pixel. **Corner horizontal axis (ho-ax)** is the horizontal ray ray drawn from left to right at a corner vertex. **Corner angle** is the angle measured anti-clockwise between a corner ho-ax and a corner diagonal at the same corner vertex. A corner pixel is identified by its associated corner angle. This method of identifi-

cation does not however mean that it is the only corner angle associated with the corner pixel.

3. Algorithms for DRC

Checks for rule violations are carried out in four directions – horizontal (left to right), vertical (top to bottom), along the 45° direction (top to bottom) and along the 135° direction (top to bottom). These are the necessary and sufficient checks to ensure reporting of all possible errors on a single layer. Further, these checks can be performed simultaneously. The algorithms can be categorised as :

- Algorithm to generate the check layers
- Algorithm to generate the error layers.

The check layer is a bitmap layer derived from the layout bitmap and identifies positions in the layout bitmap to be verified. Comparing the check layer with the layout bitmap yields the error layer. Each of these algorithms is separately discussed for single layer checks and checks for rules of layer-interactions. Algorithms for shrink operation to deal with large rules, and for identification of pixels in the layer generation algorithms have also been described in this section.

In order to identify the type of an edge or corner pixels, information of only five (top, left, right, left top corner and right top corner) out of its eight neighbours is sufficient. Also, every pixel supplies information to five of its neighbours (the neighbours in the row above the row containing the pixel are not interested in its contents).

3.1 Single layer checks

Corresponding to each direction of check, an implicit check layer is generated which is compared with the bitmap representation of the layout to generate an implicit error layer. Implicit error layers in each direction are combined to form final error layer containing error positions. For a particular direction, a check layer has runs of 1's of length specified by the design rule for that layer. These layers are implicit because they are generated by the hardware.

A horizontal Check layer has a run of 1's starting at every left edge pixel placed horizontally in left to right direction. Similarly a vertical check layer has 1's starting from the top edge pixel running down. A 135° check layer has the run of 1's starting at every left top corner pixel running towards right bottom corner. A 45° check run starts at every right top corner pixel and run towards the left bottom corner. Fig. 3.1 shows the check layers corresponding to a hypothetical layout in Fig. 2.1; the ovals mark the run of three 1's, for a rule of 3λ. Implicit check layers for spacing checks are generated in a similar manner, using the complement of the layer bitmap.

The implicit check layers described above, are used for the generation of error layer. If length of a run of 1's in the layout bitmap is $\geq$ the design rule, that run of 1's is replaced by 0's in the implicit error layer being generated marking no error. Otherwise, this run of 1's will be retained in the implicit error layer. Fig. 3.2 shows OR'ed error layers obtained from the comparison of check layer in Fig. 3.1 and layout bitmap of Fig. 2.1. Error locations for spacing checks are obtained in a similar manner using the complement of the layout bitmap.

A check for touching polygons at corner is done by generating an implicit corner error layer as follows. If a right top concave corner pixel has a top right corner pixel in the neighborhood, then the pixel is in error and is retained as a 1 in the error layer. Similarly an error is reported for the left top concave corner pixel and its top left corner neighbor.

3.2 Checks on layer interactions

Other than the width and the spacing checks on a single layer, the spacing between, the overlap of and the extension beyond overlaps of polygons on separate layers, have to be checked. These checks can be converted to spacing/width checks on a new layer; the new layer being generated by carrying out boolean operations on the interacting masks.

Polygons on different layers will share a common edge in the vertical or horizontal direction if the spacing between them is zero, or if the extension beyond overlaps is zero. In this case, direct implementation of the single layer width/spacing checks on the newly generated layer will fail, and error pixels will not be generated at these positions even if the rule specifies a width/spacing of $m > 0$. To overcome this drawback, error layers are also generated using one of the component layers of the new layer.

Error generation in this manner implies that a run of 1's can be present in the check layer with no corresponding run of 1's in the layer being checked, at all points where polygons of the component layers share an edge. Therefore, the generation of the error pixel at this point will require an addition of a clause to the error generation algorithm described for single layer checks. Details of the complete error generation algorithm for multi-layer checks can be found in [1,2].

The length of the maximum rule that can be checked is limited. To carry out checks for rules longer than this, a **shrink operation** is used for the purpose. The shrink operation works in the following manner: For a given pixel, if any of its left, top or left-top diagonal neighbour is white, the pixel is made white. The principle underlying this operation is, if a check is carried out, using a rule of length of m, on a layout which is obtained by shrinking the original layout by x in the horizontal, vertical and diagonal directions, it is equivalent to checking with a rule of length (m+x) on the original layout.

4. Special Purpose Architecture

A special pupose architecture for DRC realizes the above algorithm in hardware with its input being a bitmap of the layout. It must be noted that for a com-

plete check, hardware to perform width check is sufficient because a space check on a layout is essentially a width check performed on the complement of the layout bitmap.

The architecture of the "Design Rule Checker Chip", as we shall call it, consists essentially of a number of processing elements (PE), connected together in an array. Fig. 4.1 shows a block diagram of the DRCC with 4 PEs connected in a 2 x 2 array. Within a chip, the PEs are connected (shown by the Conn. Bus in Fig. 4.1) to generate check layers and error layers, for only two directions. These two directions can be either the horizontal and the 135 directions or the vertical and the 45 directions. The chip is programmed accordingly, to select appropriate check layer directions. Since the connection between PEs is hardwired, the programming essentially alters the manner in which the data is loaded into the PE array. A complete check can be done on two sets of DRCC arrays – one set carrying out DRC along one direction-pair, and other set along the other.

Each of the PEs can be thought to be made up of the following modules:

4.1 The Cell : The cell stores the bitmap layer, performs boolean operations on single layers or between two layers, shrink operation and generates control signal for the plane module. The cell is essentially an accumulator with two bit storage. The main components of the cell includes registers to store the bitmap data, a boolean function generator which performs logical operations on its input data, rule aligners to generate control signals for the plane modules, necessary hardware to combine neighbour information for use in shrink operation. Rule aligners are combinational gates, which use the contents of 'A' register of a PE and 'A' registers of its neighbor PEs, and determines if the pixel in 'A' register of the PE lies at a position corresponding to the beginning of the run of 1's in the check layer. The output of rule aligner is used by the plane modules within the PE at the time of generation of the respective implicit check layers.

4.2 The Planes : The planes form the heart of the hardware since and perform the functions of implicit check layer generation and error layer generation.

Each plane is made up of modules called plane modules (PMs). The PMs (both check and error layer generation) are similar for all the four directional checks. The manner of interconnection between PMs on different PEs determines the check they perform. The check and error layers are generated in two directions. This is due to the fact that one plane serves the purpose in horizontal and vertical directions while the other in 135^{o} and 45^{o} directions. Hence there are only two planes within a DRCC. These planes will be referred to as the horizontal and the 135 degree direction planes;

The horizontal check layer is generated by the horizontal check layer plane which is constructed by connecting the input conn bus of a check layer PM and the output conn bus of the check layer PM on the left. This plane is controlled by horizontal rule aligner signals. In the similar way, the 135^{o} check layer is generated by the 135^{o} check layer plane which is constructed by connecting the input conn bus of a check layer PM to the output conn bus of the left-top diagonal check layer PM. This plane is controlled by 135^{o} rule aligner signals. The Error planes are also constructed in a similar manner.

Check layer PM has inputs from the rule aligner, the rule registers and the previous implicit check layer PM. It has an output, OUT which is fed to the error layer PM. The logic state of the output OUT decides the nature of the implicit layer pixel (black or white) to which the PE corresponds.

Logic for the check layer PM ensures 0 logic level at OUT in absence of an active aligner control signal, either from its previous PM or from its own associated rule aligner.

Error Layer PM takes inputs from rule aligner, OUT signal of check layer PM, output of the boolean function generator in the PE and output of previous error layer PM. One of its outputs is connected to the next error layer PM and other output feeds Cell in the PE. The errors generated in the error layer PMs are combined together along with the corners-touching error generator to yield the final error layer pixel.

The rule register consists of a 4 bit storage, to store the design rule (maximum length permissible is 4 λ). Every DRCC has one rule register. The rule register is loaded in two cycles, two bits at a time. The output of the rule register is the rule bus (consisting of RULE[1], RULE[2], RULE[3] and RULE[4] signals). The rule register inputs specify the length of the run of 1's in the rule.

Two control lines, CNTL-A and CNTL-B are decoded by the command decoder to generate the required control signals within the chip. Address decoding is based on the observation that a 2×2 DRCC has 4 cells and only 2 data lines. Hence during a read or write operation, 2 cells will be selected, and the other 2 will not be selected. The breakup into 2 groups, of two cells each, is done by a row select line. Further, since the DRCC resides in a SIMD environment, where many such chips are present, an additional chip select signal is required. As the design rule is a constant, the loading of rule register is done simulataneusly independent of chip select.

A complete check is carried out only when all the 4 error layers have been generated. To do this, 2 DRCCs are required. Bitmaps are loaded simultaneously into both. The error outputs are wire ORed to generate the error layer. The 4 error layers can be categorized into 2 groups, (i) Horizontal and 135^{o} error layers and (ii) Vertical and 45^{o} error layers. The Data Traffic Path handler decides which of these two groups is being generated in that particular DRCC. If VT/HZ pin is at logic level one, then second group error layers are generated, otherwise first group error layers are generated. The VT/HZ pin controls a circuit which does the job of

rotating the layout by 90°, at the time of loading and error layer at the time of reading.

5. Conclusions

In this paper we proposed algorithm for performing DRC on a bitmapped layout and presented special purpose architecture for its implementation. The architecture allows configuring DRC chips for partitions of any desired size in a modular fashion. For lager partitions, test time can be reduced as the verification time is independent of partitioning. The algorithm checks only Manhattan geometries. A design of full custom hardware implementation of the proposed algorithm is currently being persued which will allow checks on larger partitions.

References

[1] Bhat, N B and Nandy, S K "New Algorithms for Hardware Acceleration of DRC",Proc. 2nd Int. Workshop on VLSI Design, Bangalore, India, 1988, pp 382-413

[2] Bhat, N B and Nandy, S K "Special Purpose Architecture for Accelerating Bitmap DRC", Tech. Rep., Lab. for CAD, Indian Institute of Science, Bangalore, November 1988.

[3] Blank, T "A Survey of Hardware Accelerators used in CAD", IEEE Design & Test of Computers, Vol. 1, No. 3, Aug 1984 pp. 21- 39.

[4] Blank, T, Stefik, M and vanCleemput, W "A Parallel Bit Map Processor Architecture for DA Algorithms", Proc. 18th Des. Aut. Conf., 1981, pp. 837-845.

[5] Seiler, L "A Hardware Assisted Design Rule Check Architecture", Proc. 19th Des. Aut. Conf., 1982, pp. 232-238.

Acknowledgements

The authors wish to thank Mr. Rajat Moona for his suggestions on implementing certain parts of the DRC accelerator, and in preparation of camera ready form of this document.

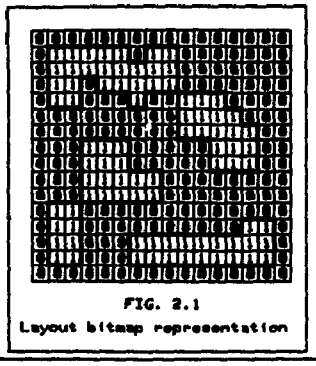

FIG. 2.1
Layout bitmap representation

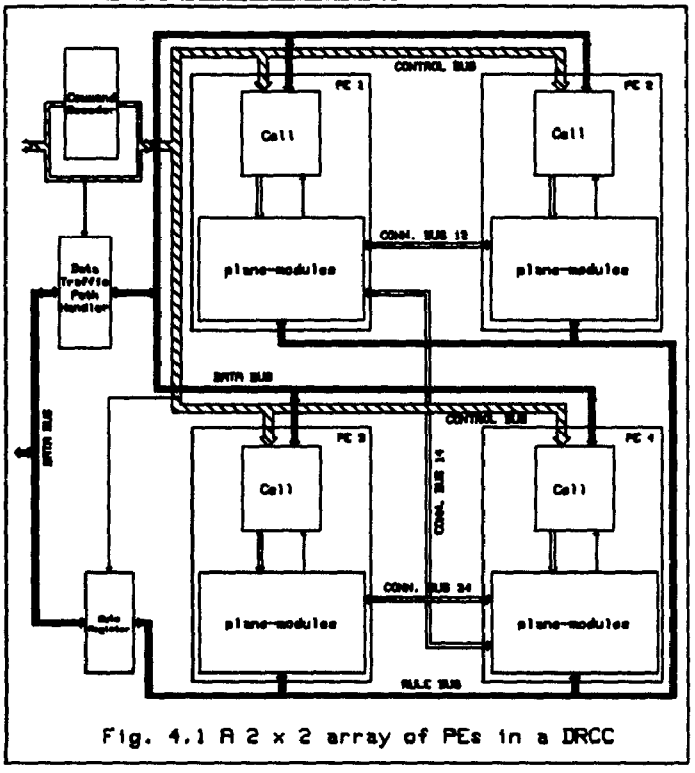

Fig. 4.1 A 2 x 2 array of PEs in a DRCC

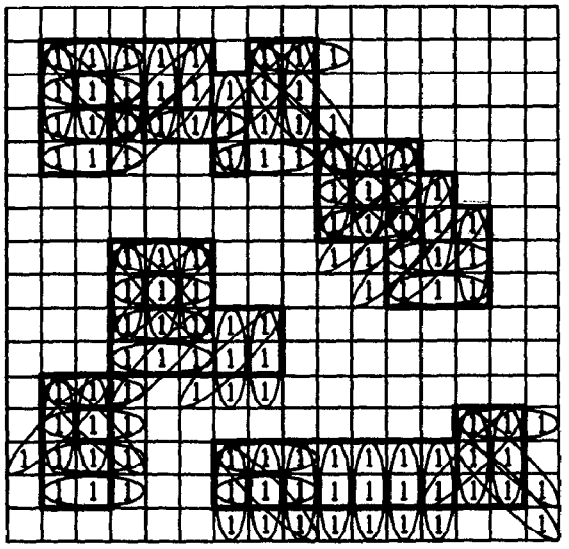

FIG. 3.1 Check Layers
(Design rule =3)
(The 0s have not been shown)

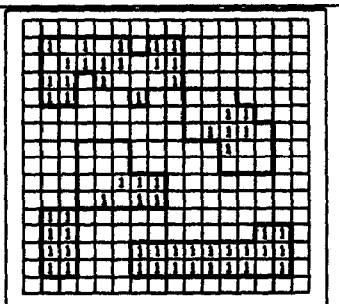

FIG. 3.2 ORed Error Layers