

Problem 9: Area Filling

Source: Cadence Taiwan, Inc.

Dec. 20, 2000

I. Introduction

In order to control the manufacturing variation due to *chemical-mechanical polishing* (CMP), the pattern density distribution of layout features on a layer must be uniform. To change local pattern density, and thus ensure post-CMP planarization, “filling” patterns are inserted to the layout. The procedure is called “Area Filling”. It is sometimes called “tiling” because the filling patterns typically are small rectangles of similar shape to minimize coupling. There are several design rules associated with the filling patterns defined by the foundries, such as the filling pattern size, the distance between two adjacent filling patterns, and the distance between the filling patterns and original layout shapes. For a simple situation, the filling patterns can be placed in array type with alignment and repetition in both X and Y directions separated by a specified distance (normally the minimum distance between the filling patterns) – see Figure 3a. While for a complicated process, in order to minimize the coupling effect and improve the quality of layout pattern distribution, staggered type of filling patterns are required – see Figure 3b.

II. Input/Output Specification

As Figure 1 shows, the tool should receive two input files (*input1* and *input2*) and generates a set of output files described as follows.

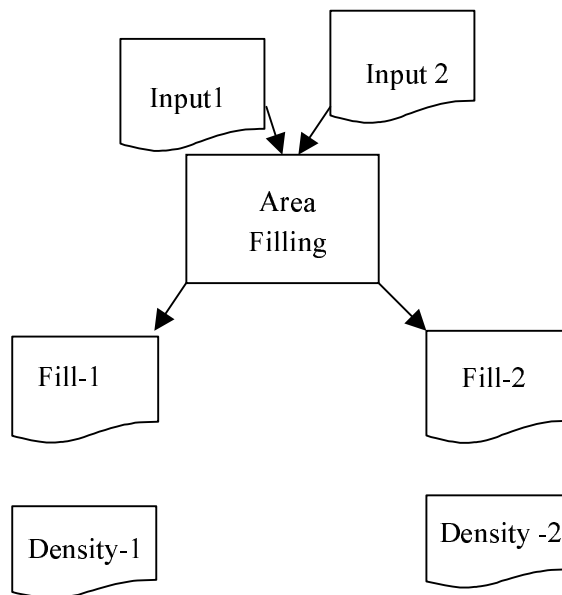


Figure 1. Inputs and outputs

Input Format:

File *input1* described in CIF (Caltech Intermediate Form) format consists of layout data which contain (1) a BOX defining the boundary for area filling and (2) a set of shapes defining the exclusion zones that the filling patterns can not be inserted (butting to the zones is OK, but no overlapping is allowed).

The detailed CIF format can be found in the book of “Introduction to VLSI Systems” written by Carver Mead and Lynn Conway. A simple description of the CIF format can also be found at the end of this article. To simplify the problem, we assume

- (1) the layout is flattened – i.e. there is no calling symbols (cell instances) in the input CIF file.
- (2) the exclusion zones are all rectilinear shapes in Box, or Polygon formats represented with Layer 1, and
- (3) the design boundary is represented in a single box command with Layer2.

For example, Figure 2 shows the input shapes of the layout and its the corresponding CIF file.

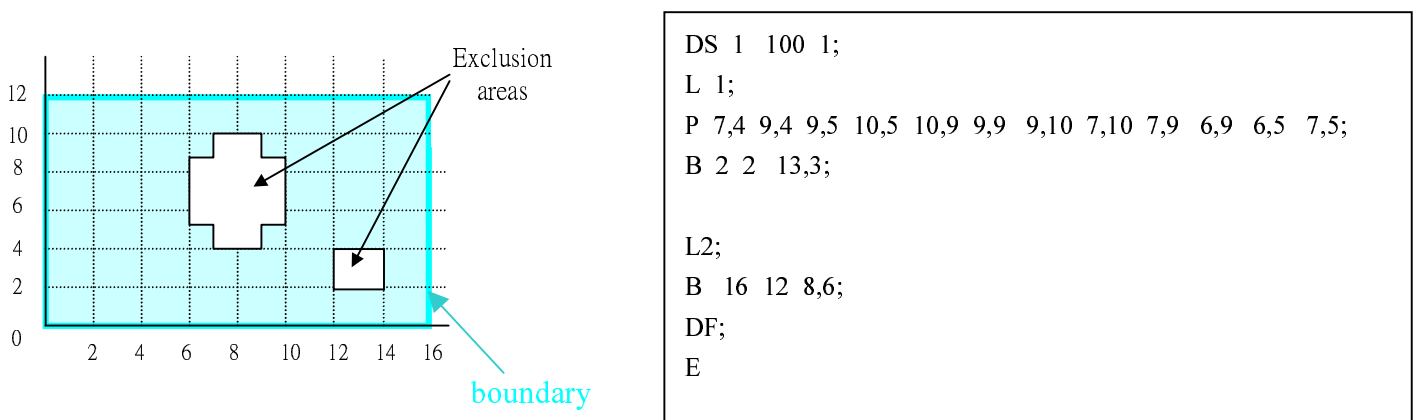


Figure 2. Input example

File *input2* is a fixed format of ASCII file containing information for the filling patterns:

- Line1: Filling patterns' dimensions: a set of width, length pairs to specify the dimensions of rectangles, like 1,2 1,3 1,4 1,5
- Line2: Minimum spacing between the filling patterns, like 1.0
- Line3: Minimum density requirement, like 0.3
- Line4: Offset's resolution. This is the x/y offset for staggered filling patterns (see Figure 3). The program should choose a suitable non-zero offset value so that the filling patterns can be represented as array-typed or group of instances. The value of the offset should be greater than 0.0 and equal or less than $\frac{1}{2}$ of the minimum spacing between filling patterns. For example, if the minimum spacing is 1.0, and the resolution is 0.05, then the program can use 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45 or 0.5 as the offset value.

Example of the input2:

```
1,2 1,3 1,4 1,5
1.0
0.3
0.05
```

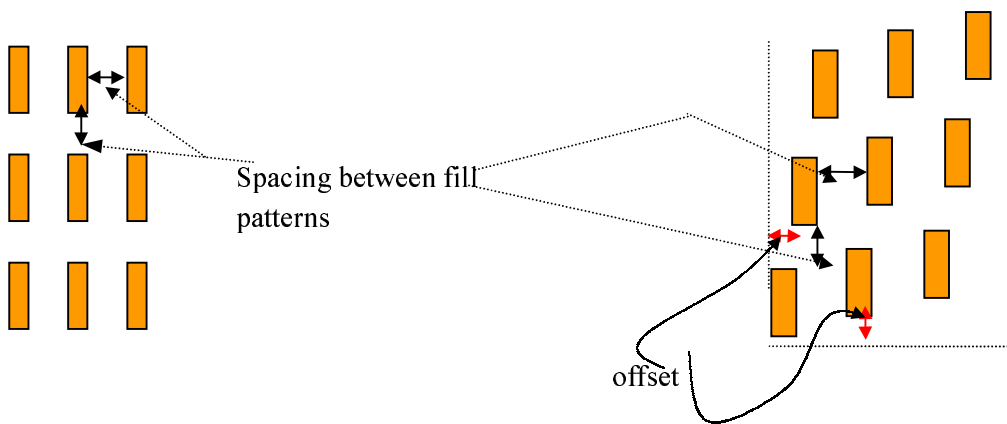


Figure 3a. Offset=0

Figure 3b. Non-zero valued offset

Output Format:

Two sets of outputs are produced. Each set contains

- (1) the filling patterns' layout description in CIF format and
- (2) the total density of the layout, i.e.

(Total Area of the filling patterns + Total area of exclusions) / Total Area of the boundary.

The first set is the result considering area filling without offset, i.e. offset=0. The second set is the result considering offset, i.e. offset != 0. For both results, the filling patterns can be represented in flattened shapes, instances with several levels of hierarchies, arrays, etc.

III. Problem Statement

The problem is to take the geometric layout data contained in *input1* and perform area filling using the filling patterns defined in *input2* such that the requirements specified in *input2* are satisfied and the required output files are produced. There will be 3 test cases with different degrees of difficulty provided for this problem. The program needs to consider a general solution for the filling pattern generation, especially for the staggered filling patterns. The inserted filling patterns need to follow the spacing rules and meet the minimum density requirements. They should be also distributed as evenly as possible, and the amount of output layout data records needs to be minimized in order to make the output file size as small as possible.

IV. Language/Platform

1. Language: C or C++.
2. Platform: No constraints.

V. Evaluation

The score will be given based on the following criteria with priority in order,

- (1) The total density of the layout must meet the requirement, but it should be made as higher as possible.
- (2) The distribution degree of the filling patterns – how well the filling patterns are distributed. Dracula COVERAGE command can be employed to check the distribution quality.

- (3) The file-size of the output layout data in CIF format. The program should make it as small as possible by utilizing array, hierarchical representations.
- (4) The run time – performance is also a key factor.

VI. Questions

Please report any question regarding this problem to cad@cis.nctu.edu.tw with the email subject “CAD Contest: Problem 9.” Your question(s) will be answered in two weeks, and the Q&A’s will be posted at the contest web site

References

- [1] A. B. Kahng, G. Robins, A. Singh, and A. Zelikovsky, “Filling Algorithms and Analyses for Layout Density Control,” *IEEE Trans. CAD of ICs and Systems*, Vol 18, No. 4, Pages 445-462, Apr. 1999.
- [2] Yu Chen, Andrew B. Kahng, Gabriel Robins, and Alexander Zelikovsky, “Practical Iterated Fill Synthesis for CMP Uniformity,” In Proc. *Design Automation Conference*. Pages 671-674, 2000.
- [3] Ruiqi Tian, D.F. Wong, and Robert Boone, “Model-Based Dummy Feature Placement for Oxide Chemical-Mechanical Polishing Manufacturability,” In Proc. *Design Automation Conference*. Pages 667-670, 2000.

Appendix - CIF records

A CIF file is composed of a sequence of characters in a limited character set. The file contains a list of commands, followed by an end marker; the commands are separated with semicolons.

| Command | Form |
|---|--|
| Polygon with path | P path |
| Box with length, width, center and direction (direction defaults to (1,0) if omitted) | B integer integer point point |
| Wire with width and path | W integer path |
| Layer specification | L shortname |
| Start symbol definition with index, a, b (a and b both default to 1 if omitted) | DS integer integer integer |
| Finish symbol definition | DF |
| Call Symbol | C integer transformation |
| User extension | Digit user Text |
| Comments with arbitrary text | (comment Text) |
| End marker | E |
| Array Option: 0A symbolid column row dx dy transformation (note: this is user defined syntax for array representaion) | 0A integer integer integer integer integer transformation |

A more formal definition of the syntax is given below.

cifFile = { {blank} [command] semi } endCommand {blank}.

command = primCommand | defDeleteCommand | defStartCommand semi { {blank} }
[primCommand] semi defFinishCommand.

| | |
|----------------------|---|
| primCommand | = polygonCommand boxCommand roundFlashCommand wireCommand layerCommand callCommand userExtensionCommand commentCommand. |
| polygonCommand | = "P" path. |
| boxCommand | = "B" integer sep integer sep point [sep point]. |
| roundFlashCommand | = "R" integer sep point. |
| wireCommand | = "W" integer sep path. |
| layerCommand | = "L" {blank} shortname. |
| defStartCommand | = "D" {blank} "S" integer [sep integer sep integer]. |
| defFinishCommand | = "D" {blank} "F". |
| defDeleteCommand | = "D" {blank} "D" integer. |
| callCommand | = "C" integer transformation. |
| userExtensionCommand | = digit userText. |
| commentCommand | = "(" comment Text ")". |
| endCommand | = "E". |
| transformation | = { {blank} ("T" point "M" {blank} "X" "M" {blank} "Y" "R" point) }. |
| path | = point {sep point}. |
| point | = sinteger sep sinteger. |
| sinteger | = {sep} ["-"] integerD. |
| integer | = {sep} integerD. |
| integerD | = digit {digit}. |
| shortname | = c[c][c][c]. |
| c | = digit upperChar. |
| userText | = {userChar}. |
| commentText | = {commentChar} commentText "(" commentText ")" commentText. |
| semi | = {blank} ";" {blank}. |
| sep | = upperChar blank. |
| digit | = "0" "1" "2" "3" "4" "5" "6" "7" "8" "9". |
| upperChar | = "A" "B" "C" ... "Z". |
| blank | = any ASCII character except digit, upperChar, "-", "(", ",", or ";". |
| userChar | = any ASCII character except ";". |
| commentText | = any ASCII character except "(" or ")". |