VLSI Design for Manufacturability

Project (Due: 2025/5/29 23:59)

Introduction

In modern VLSI designs, IR drop mitigation is critical for ensuring circuit reliability and performance, especially as technology nodes continue to scale down. Power staple insertion has emerged as a key technique for strengthening power and ground networks by creating additional metal connections (staples) between adjacent power or ground rails. However, the effectiveness of staple insertion heavily depends on the availability of empty vertical spaces between rows after placement.

This project, based on [1], focuses on placement refinement and power staple insertion. Given an initial placement as shown in Fig. 1(a), find a refined non-overlapping placement along with staple insertions (e.g. Fig. 1(b)) by horizontally shifting and flipping cells such that the total number of staples inserted is maximized subject to the following constraints.

1. Same-row constraint: Each cell must stay within its original row and cannot be moved to another row.
2. Maximum displacement constraint: Each cell can only be moved within (≤) a specified maximum displacement. (You may change the order of cells as long as the maximum displacement constraint is not violated.)
3. Staggering constraint: The refined placement must not introduce any staggering pattern. For example, the refined placement shown in Fig. 1(c) introduces two staggering patterns which would be infeasible.
4. Alignment constraint: The left boundary of every cell and staple must align with a placement site, and the bottom boundary must align with a placement row.
5. Staple balance constraint: The number of VDD-to-VDD staples and VSS-to-VSS staples must be balanced, such that the ratio between the larger and the smaller count does not exceed (≤) 1.1.

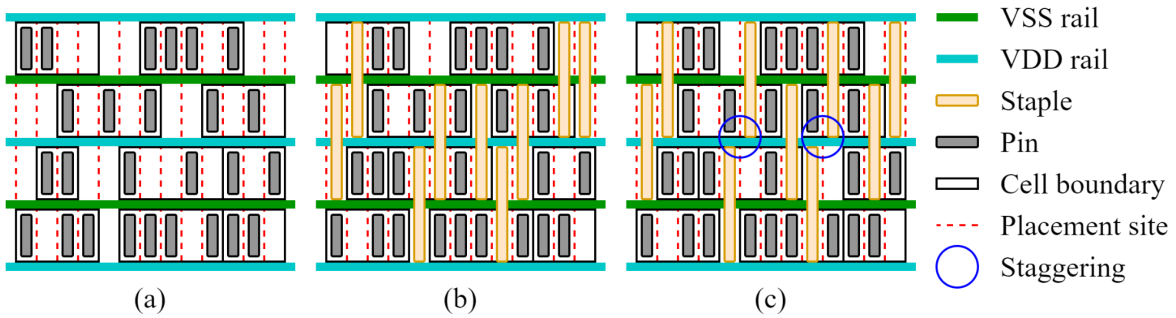


Figure 1. (a) An initial placement with four placement rows and nine cells. (b) A refined placement with nine inserted staples. (c) A refined placement with nine inserted staples. However, highlighted are two staggering patterns.

Input Format

There are three parts in the input file.

1. The first part consists of four lines.
    a. The first line lists the x-coordinate of the chip left boundary, the y-coordinate of the chip bottom boundary, the x-coordinate of the chip right boundary, and the y-coordinate of the chip top boundary.
    b. The second line lists the number of placement rows, the height of every placement row, and the width of each placement site.
    c. The third line lists the number of cell types.
    d. The fourth line lists the number of cells.

    Format:
    <x-coord. of left chip boundary> <y-coord. of bottom chip boundary>
    <x-coord. of right chip boundary> <y-coord. of top chip boundary>
    <#placement rows> <height of a row> <width of a placement site>
    <#cell types>
    <#cells>

    A sample of the first part is given below.

```
0 0 120064 121344
78 1536 128
10
5000
```

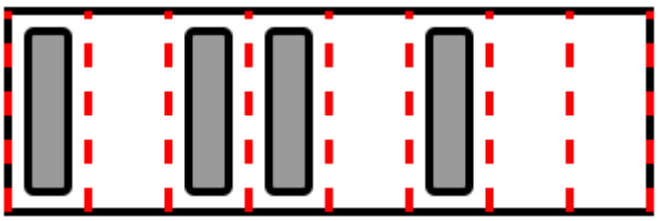
2. The second part consists of #cell types lines. The information of each cell type is given by a sequence of numbers. The first number is the cell type index, and the second and third numbers represent the width and height of the cell type, respectively. The remaining numbers indicate the indexes of the placement sites occupied by pins within the cell (the first site from the left boundary of the cell has an index of 0).

    Format:
    <cell type index> <width of the cell type> <height of the cell type> <pin site 1> <pin site 2> …

    A sample of the second part is given below.

```
0 1024 1536 0 2 4
1 1152 1536 1 4 5 6
2 512 1536 0 2
…
```

For example, when the width of a placement site is 128, a cell type defined as “0 1024 1536 0 2 3 5” is shown below.



3. The third part consists of #cells lines. The information of each cell is given by a separate line listing the cell index, the cell type index, the x-coordinate of the cell left boundary, the y-coordinate of the cell bottom boundary, and the maximum allowed displacement of the cell (initially, all cells are not flipped).

   Format:
   <cell index> <cell type index> <x-cord. of the cell left boundary> <y-cord. of the cell bottom boundary> <maximum disp. of the cell>

   A sample of the third part is given below.

```
0 2 0 0 1280
1 0 0 1536 768
2 1 2048 0 1024
…
```

   The maximum displacement of cells is no more than 7 placement sites in this project.

Output Format

There are two parts in the output file.

1. The first part consists of #cells lines. The information of each cell is given by a separate line listing the cell index, the x-coordinate of the cell left boundary in the refined placement, the y-coordinate of the cell bottom boundary in the refined placement, and a flag indicating whether the cell is horizontally flipped (0 indicates non-flipped and 1 indicates flipped).

   Format:
   <cell index> <x-cord. of the cell left boundary in the refined placement> <y-cord. of the cell bottom boundary in the refined placement> <flipping flag>

A sample of the first part is given below.

```
0 128 0 0
1 256 1536 0
2 2048 0 1
…
```

2. The second part consists of N lines where N is the number of staples you inserted in the refined placement. The information of each staple is given by a separate line listing the x-coordinate of the staple left boundary and the y-coordinate of the staple bottom boundary.

   Format:
   <x-cord. of the staple left boundary> <y-cord. of the staple bottom boundary>

   A sample of the second part is given below.

```
0 0
128 3072
…
```

Project Submission and Makefile Requirement

The source codes should be uploaded to eeclass. Please include a Makefile for compiling your codes. In addition, upload a report describing the details of your approach.

Name the executable file “Staple_Insertion” and make sure your program can be executed by running the command:
$./Staple_Insertion <input file path> <output file path>

like the one below

```
./Staple_Insertion ./input/bm1.txt ./output/bm1.txt
```

All codes should be compiled by running the command:
$make
You have to delete the object file and executable file by running the command:
$make clean

Environment and Execution

1. Language: C/C++ (gcc version 9.4.0)
2. Platform: Linux (Ubuntu 18.04.6 LTS)

Evaluation

1. For each benchmark, if your result violates any constraints, then the quality score on that benchmark will be 0.
2. If your program takes more than 600 seconds to generate a result or uses more than 16GB of memory, it fails on that benchmark.
3. Any plagiarism will result in a 0 grade for the project.
4. If we cannot compile your source code or we cannot execute your program by the command mentioned above, then the quality score for all benchmarks will be 0.

Grading

- 30%: The completeness of your report.
- 70%: The solution quality (hidden benchmarks included)
    - For each benchmark, the quality score is computed based on the #staples of your solution compared to other students when your solution is valid. Here is the equation for score calculation.

$$60 + 40 \times \frac{your\ \#staples - smallest\ \#staples}{largest\ \#staples - smallest\ \#staples}$$

Reference

[1] Y. -J. Xie, K. -Y. Chen and W. -K. Mak, "Manufacturing-Aware Power Staple Insertion Optimization by Enhanced Multi-Row Detailed Placement Refinement," 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, 2021, pp. 872-877.