Department of Computer Science and Information Engineering

CS6135 VLSI Physical Design Automation Homework 4: Analog Device Placement Considering Symmetry Constraints

黃偉祥 X1136010

Lecturer: Ting-Chi Wang

A report of CS6135 course Homework 4
Institute of Computer Science and Information Engineering

May 8, 2025

Declaration

I, Wei-Xiang Wong, of the Department of Computer Science and Information Engineering, National Tsing Hua University, confirm that this is my own work, figures, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalized accordingly.

I give my consent to having a copy of my report shared with future students as an example.

Wei-Xiang Wong May 8, 2025

Contents

1 About

1.1 Introduction

Implement an analog device placer considering symmetry constraints, using the benchmark circuits published in the 2009 IEEE TCAD paper entitled **Analog Placement Based on Symmetry-Island Formulation** by Lin et al. (2009)

1.2 Problem Description

1. **Input:**
   - A set B of rectangular hard blocks corresponding to different analog devices, where each block b_i in B has its width and height.
   - A set of symmetry constraints containing one or multiple symmetry groups. Each symmetry group consists of more than one symmetry pairs, (b_i, b_i'), and/or self-symmetric device.
2. **Output:**
   - The coordinates (x_i, y_i) of the lower-left corner of each block b_i, as well as the rotation status (1 for rotated, and 0 for unrotated).
3. **Objective:**
   - By assuming each block can be rotated by 90 degrees, the objective is to minimize the total area of the analog device placement result. The coordinate of each block in the placement result should be positive. Total area is calculated by the minimum rectangle bounding all the blocks with the lower-left corner starting from $(0, 0)$. Furthermore, the placement result is subject to the following constraints.

   Constraints:

   (a) **Symmetry constraint:** Each symmetry group must form a symmetric placement with respect to a vertical/horizontal symmetry axis.

   (b) **Non-overlapping constraint:** No two blocks overlap with each other.

1.3 Input File

1. The `.txt` **file:**

   The `.txt` file specifies the information about the analog devices.

```
NumHardBlocks 10
// NumHardBlocks number of hard blocks
HardBlock hb0 12 331
// HardBlock block name width height

⋮

NumSymGroups 2
// NumSymGroups number of symmetry groups
SymGroup sg0 2
// SymGroup group name number of pair- and self-symmetry blocks
SymPair sp0 sp1
// SymPair block name1 block name2
SymSelf ss0
// SymSelf block name
SymGroup sg1 3
SymPair sp2 sp3
SymPair sp4 sp5

⋮
```

1.4 Output File

1. The `.out` **file:**

   The `.out` file specifies the analog device placement result including:

   - The total area of the bounding rectangle.
   - The total number of the analog devices.
   - The coordinates of the lower-left corner of each analog device block with/without rotation.

```
Area 11324900
NumHardBlocks 49
hb0 0 0 1
// block name lower-left corner coordinates (x,y) rotated
hb1 0 1708 0
// block name lower-left corner coordinates (x,y) unrotated

⋮
```

1.5 Compile & Execute

1. Go into directory **src/**, enter **make** to compile my program and generate the executable file, called **hw4**, which will be in directory **bin/**.

2. Go into directory **src/**, enter **make clean** to delete the executable file.

3. Use the following command format to run my program:

$ ./hw4 *.txt *.out

E.g.:

$ ./hw4 ../testcase/public1 txt ../testcase/public1 out

2 Result

Result of **HW4_grading.sh**

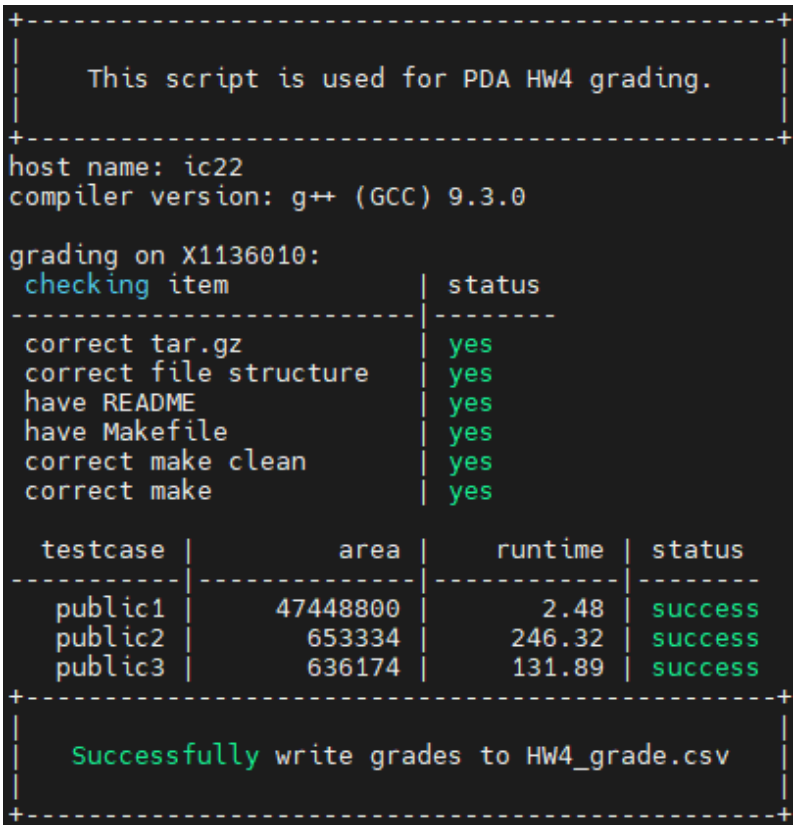

Figure 2.1: Result of HW4_grading.sh

testcase	area	runtime(s)
public1	47448800	2.48
public2	653334	246.32
public3	636174	131.89

Table 2.1: Result of HW4_grading.sh

2.1 public1 result

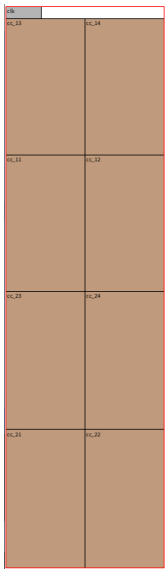

Figure 2.2: Result of public1

2.2 public2 result

Figure 2.3: Result of public2

2.3 public3 result

Figure 2.4: Result of public3

3 Implementation

3.1 B*-tree

I add some features to the template **B*-tree**:

1. **symaxis** : To record the symmetry axis of the symmetry group.
2. ***parent** : To record the parent Node.
3. **method** : To record which method used to build the symmetry island.
4. **rotate** : To record is the block rotate.
5. **is_sg** : To record is the block a symmetry group.
6. **name** : To record the Node's name.
7. **setRotate()** : To rotate the Node.

```
    T x, y;
    T width, height;
    T symaxis;
    Node *lchild, *rchild, *parent;
    int method=0;
    bool rotate;
    bool is_sg;
    std::string name;

    void setRotate(){
        rotate = !rotate;
        std::swap(width,height);
    }
```

Listing 3.1: Addition to Node

3.2 Input parser

Store the input information using a class **InputParser** with the data structure show as **Listing 3.2: Input Parser**

```
struct HardBlock {
    std::string name;
    int64_t width;
    int64_t height;
    int64_t x=0;
    int64_t y=0;
```

```
7       bool rotate=false;
8       int64_t W() const { return rotate ? height : width; }
9       int64_t H() const { return rotate ? width : height; }
10  };
11
12  struct SymPair {
13      std::string block1;
14      std::string block2;
15  };
16
17  struct SymSelf {
18      std::string block;
19  };
20
21  struct SymGroup {
22      std::string name;
23      std::vector<SymPair> pairs;
24      std::vector<SymSelf> selfs;
25  };
26
27  class InputParser {
28  ...
29  ...
30  ...
31  }
```

Listing 3.2: Input Parser

3.3 Methods of building symmetry group

3.3.1 Method 1: Max row first

First, sort all the block in the symmetry group by its **width**(if height $>$ width will rotate), and get the maximum width to be the bound of each row. Then put the block by descending order of the width row by row, if the block excess the bound(Figure 3.1) then put into the next row(Figure 3.2). **Red** dot line is the symmetry axis, **green** dot line is the bound of each row.

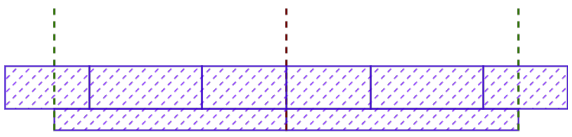

Figure 3.1: Block excess the bound

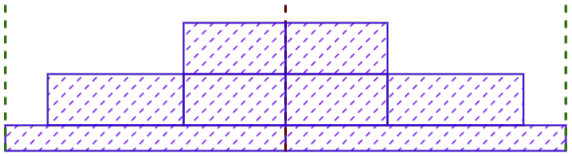

Figure 3.2: Put in next row

3.3.2 Method 2: Stack the height

First sort the block through its **height**(if width $>$ height will rotate), and get the maximum width to be the width of the symmetry group.

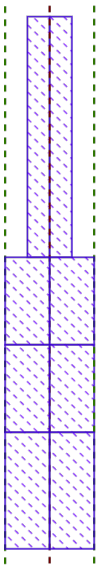

Figure 3.3: Method 2: stack the height

3.4 Construct nodes

For each symmetry group, use method 1 and method 2 to build and compare, if method 1 have smaller area the use method 1 else method 2. For each normal block just see as a node.

3.5 Initial solution

For each nodes, put the next node be the right child of the node until end, so the block will keep stacking up. (Using B*-tree)

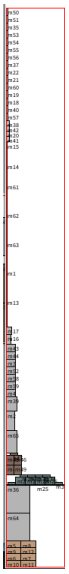

Figure 3.4: public2.txt initial solution example

3.6 Perturbation

3.6.1 Op1

Rotate a node, same as Op1 in B*-tree perturbation.

3.6.2 Op2

Delete & Insert a node, same as Op2 in B*-tree perturbation.

3.6.3 Op3

Swap 2 nodes, same as Op3 in B*-tree perturbation.

3.6.4 Op4

Move a single node, this operation will only move a single node(with maximum 1 child) to be left or right child of a random node iff the random node has only 1 child.

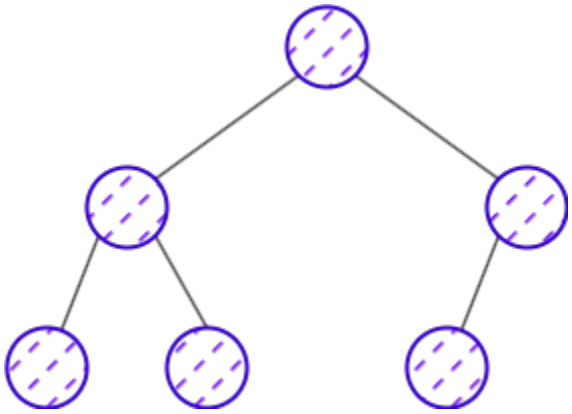

Figure 3.5: Before Op4

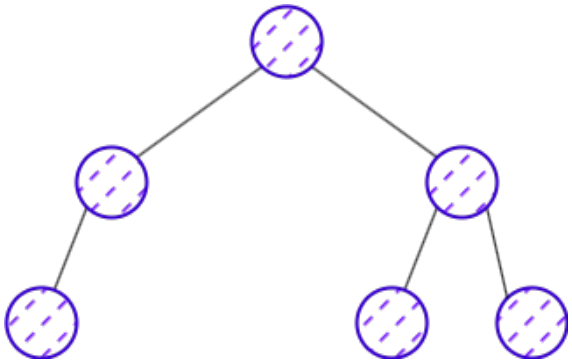

Figure 3.6: After Op4

3.6.5 Op5

Same as Op4 but will rotate the shape of the node when moving.

4 Tricks

Use **public2** as example.

4.1 Stacking up

Instead of build a binary tree as initial, stack the tree can have a smaller size. $693000- > 667184$

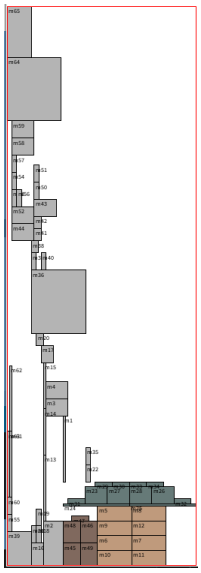

Figure 4.1: Complete binary tree as initial

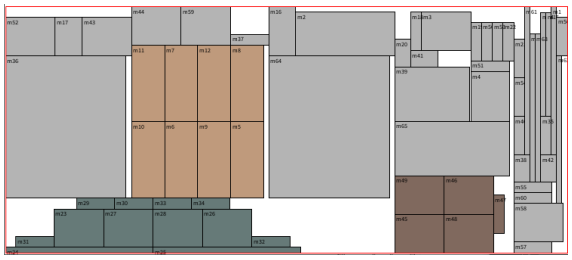

Figure 4.2: Complete binary tree as initial(Final result)

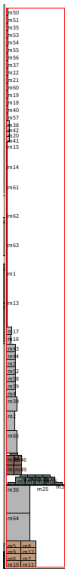

Figure 4.3: Stacking up as initial

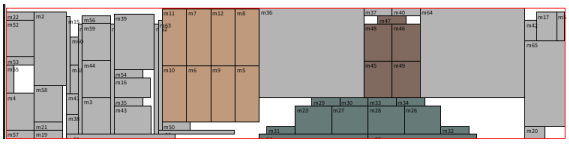

Figure 4.4: Stacking up as initial(Final result)

4.2 Extra Op

Introducing extra move of Op4 & Op5 to let the block can move more flexible and have better result. $667184 -> 653334$

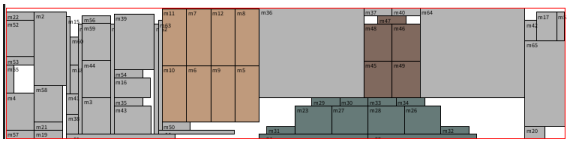

Figure 4.5: No extra Ops(Final result)

Figure 4.6: With extra Ops(Final result)

5 Conclusion

1. **Symmetry constraints**, this is hard when considering the symmetry constraints, in the end I realize that building a symmetry island will make the problem easier. I build the symmetry group as a single node and perform perturbation as normal B*-tree and finally pass the baseline.
2. **Different** patterns of symmetry island can make very big impact in the final result, so I try to consider possible combination of symmetry island.
3. **Extra Ops** may not always provide better result for all cases, for **public2** extra Ops will provide better result but for **public3** will not.

References

Lin, P.-H., Chang, Y.-W. and Lin, S.-C. (2009), 'Analog placement based on symmetry-island formulation', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **28**(6), 791–804.