

# **EE6094 CAD for VLSI Design**

## **Final Project Check Point III**

Group: 3

Student ID: 108501554, 108501537, 108501023

Student name: 陳威呈, 蔡雨蓁, 李品賢

## (1) Background

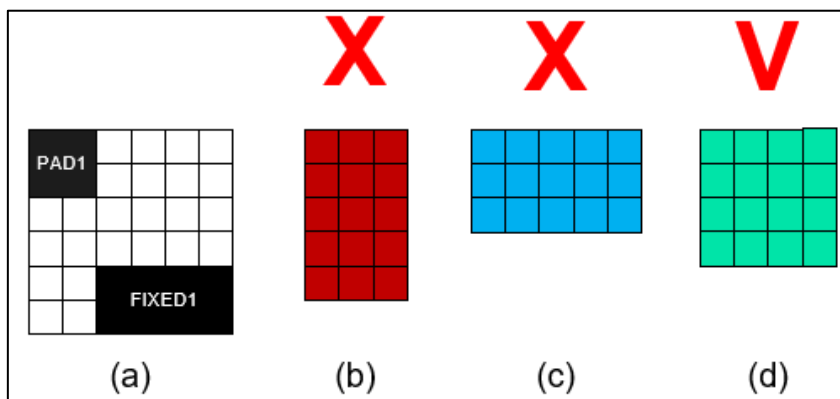
With the continuous improvement of technology, the complexity of chip design and the number of transistors are constantly increasing. In response to the increasing intricacy of chip design and transistor count, hierarchical design and IP modules have been widely used. Floorplanning, a vital element in these design approaches, plays a crucial role. It not only offers valuable insights in the early stages to help assess system architecture but also facilitate chip area estimation and prediction of delays and congestion caused by routing. Therefore, floorplanning has remained an indispensable part of VLSI circuit design and has grown even more significant.

Apart from minimizing chip area, integrated-circuit floorplanning often handles other crucial issues, such as managing fixed-outline constraints and accommodating soft modules. Dealing with these challenges requires the implementation of optimization methods and techniques to maximize design efficiency and quality.

## (2) Purpose and introduction of this problem

As the chip size is determined during the early design stage. This results in fixed-outline constraints and the conventional floorplanning methods that focus solely on minimizing area are not fully applicable in modern IC design. Instead, by considering fixed outlines in the floorplanning process, it becomes possible to strategically place modules within the fixed outline. This way, we can maximize spatial utilization and minimize wire lengths within the fixed outline. Therefore, the fixed-outline floorplanning methods are indispensable in modern IC design.

Besides, the fixed-outline floorplanning methods should have the ability to cope with soft modules. An example will be shown below to illustrate the importance of handling soft modules.



**Fig. 1** An example to illustrate the importance of handling soft modules

As shown in Fig. 1(a), the black cells within the fixed outline are occupied by fixed modules, leaving no room for placing other modules. If the floorplanning process cannot handle soft modules effectively, for a soft module with an area of 15 units, its shape can only be determined as a minimum rectangle ( $3 \times 5$  [as shown in Fig. 1(b)] or  $5 \times 3$  [as shown in Fig. 1(c)]), and it cannot be legally placed within the fixed outline. However, if the floorplanning process can

effectively handle soft modules and determine their shape as a rectangle with an area of 16 units ( $4 \times 4$  [as shown in Fig. 1(d)]), then this module can be legally placed within the fixed outline. Similarly, a soft module with an area equal to or greater than 17 units cannot be legally placed within the chip outline as a rectangle. However, if the floorplanning process can generate polygonal shapes for soft modules, soft modules with an area equal to or greater than 17 units can be legally placed within the fixed outline in a polygonal shape. Therefore, fixed-outline floorplanning must have the capability to handle soft modules in order to meet the requirements of modern IC design. The main purpose of this problem is to place soft modules in a given fixed outline and some given fixed modules.

### (3) Problem formulation

#### Given

1. The width and height of a chip outline (with the bottom-left coordinate of the chip outline as the origin (0,0))
2. A group of soft modules and their minimum required area
3. A group of fixed-shaped rectangular modules and their width, height, and bottom-left coordinates
4. The number of connections (two-pin nets) between modules (including both soft and fixed-shaped modules)

#### Goal

Implement a fixed-outline floorplanner that aims to minimize the total half-perimeter wirelength (total HPWL). Given constraints on module shapes and positions, determine the optimal shapes and positions of all soft modules.

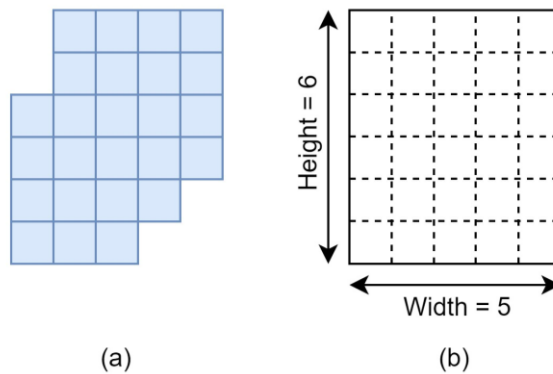
$$\text{Total HPWL} = \sum \text{Manhattan Distance} \times \text{The number of connection (between two modules)}$$

#### Restriction

1. The shape must be a simple rectilinear polygon.
  - I. Edges are parallel to the axes of the rectangular coordinate system, and corners are right angles.
  - II. No edges intersect with each other (intersect-free).
  - III. The polygon encloses only one region, and there are no holes within the region.
2. Minimum area constraint: Each soft module has a minimum-area requirement, where the area enclosed by the module's shape must be greater than this constraint.
3. The shape must satisfy an aspect ratio constraint within the range of 0.5 to 2: The aspect ratio of a soft module is the minimum ratio of height to width of the smallest enclosing rectangle for that module. The aspect ratio of a rectangle is calculated as its height divided by its width. As shown in Fig. 2, Fig. 2(a) represents a module with its shape determined, while Fig. 2(b) represents the smallest enclosing rectangle that fully covers the module in

Fig. 2(a). The aspect ratio of the module in Fig. 2(a) is calculated as the height of the rectangle in Fig. 2(b) divided by its width, resulting in a value of 1.2.

4. The shape must satisfy a rectangle ratio constraint within the range of 80% to 100%: The rectangle ratio of a soft module is calculated as the area enclosed by the module's shape divided by the area of the smallest enclosing rectangle for that module. As shown in Fig. 2, the rectangle ratio of the module in Fig. 2(a) is calculated as the area of the blue region divided by the area of the gray region in Fig. 2(b), resulting in a value of 83.3%.



**Fig. 2** An example of the outline of a soft module

#### (4) Idea modification

##### Original edition

- Slack-based SA algorithm with sequence pair representation

The simulated annealing algorithm is used to efficiently explore possible solution space. With SA algorithm, we drive it by some ingenious move types aiming at improving chip area, wirelength and soft-block aspect ratio, which is called slack-based move. Also, the new objective function is purposed to deal with the fixed-outline constraint of problem D. For the purpose of implementing this idea, the sequence pair representation is used due to its simplicity to change from current state to neighborhood state.

- Limitation

This approach entails some catastrophic drawbacks. First drawback is that it can't handle fixed modules which is a given constraint. We make this conclusion because we have read the accompanying reference paper [4] of this method [3]. This paper claims that its method handles the fixed-module placement, but how it handles fixed modules is not what we want. It defines a fixed module by giving a range restriction with its location. It totally does not conform to our problem constraint.

Secondly, the purposed method [3] to calculate LCS of sequence pair do not consider the module overlapping situation. In our problem, it needs to accept some inevitable overlapping, so the floorplan is able to success. It does not truly overlap two modules. Instead, it is the situation that the rectangle of one module is not filled up.

**New edition**

- Improvement

We design a new data structure and algorithm targeting at all given constraints of problem D. Different from the original edition, this method can address fixed-module and overlapping problem. In addition, we can easily check if each module fits the given shape and location constraints in our new algorithm through the concept of contour, which will clearly explain in the following parts.

**(5) Data structure**

1. **struct SoftModule** – indicate the information of each soft modules

**Tab. 1** Information of struct SoftModule

Data type	Name	Purpose
string	name	Records the name of a soft module
int	label	Records the label of a soft module in chip grid
int	area	Records the area of a soft module
vector <pair<int, int>>	aspect	Records the aspect of a soft module
vector <pair<int, int>>	coordinate	Records corner coordinates of a soft module

2. **struct FixedModule** – indicate the information of each fixed modules

**Tab. 2** Information of struct FixedModule

Data type	Name	Purpose
string	name	Records the name of a fixed module
int	xCoord	Records the x-axis coordinate of a fixed module
int	yCoord	Records the y-axis coordinate of a fixed module
int	width	Records the width of a fixed module
int	height	Records the height of a fixed module

3. **struct Connection** – indicate the relationship of connection between two modules

**Tab. 3** Information of struct Connection

Data type	Name	Purpose
string	module1	Records the name of a module
string	module2	Records the name of an another module
int	numNets	Records the net number between two modules

#### 4. struct Chip – indicate the aspect of the chip

**Tab. 4** Information of struct Chip

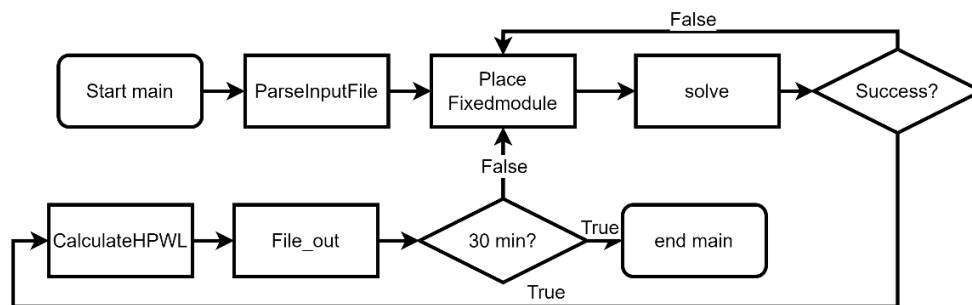
Data type	Name	Purpose
int	width	Records the width of a chip
int	height	Records the height of a chip

#### 5. Important variables

**Tab. 5** Information of important variables

Data type	Name	Purpose
vector <vector<int>>	Grid	Records the grid of a chip
int	x_pointer	Points to the x-location of the contour in order to indicate the location of current-placing module
vector<int>	x_contour	Records the x-axis contour to indicate the current module placement

#### (6) High-level flow chart of our program



**Fig. 3** High-level flow chart of our program

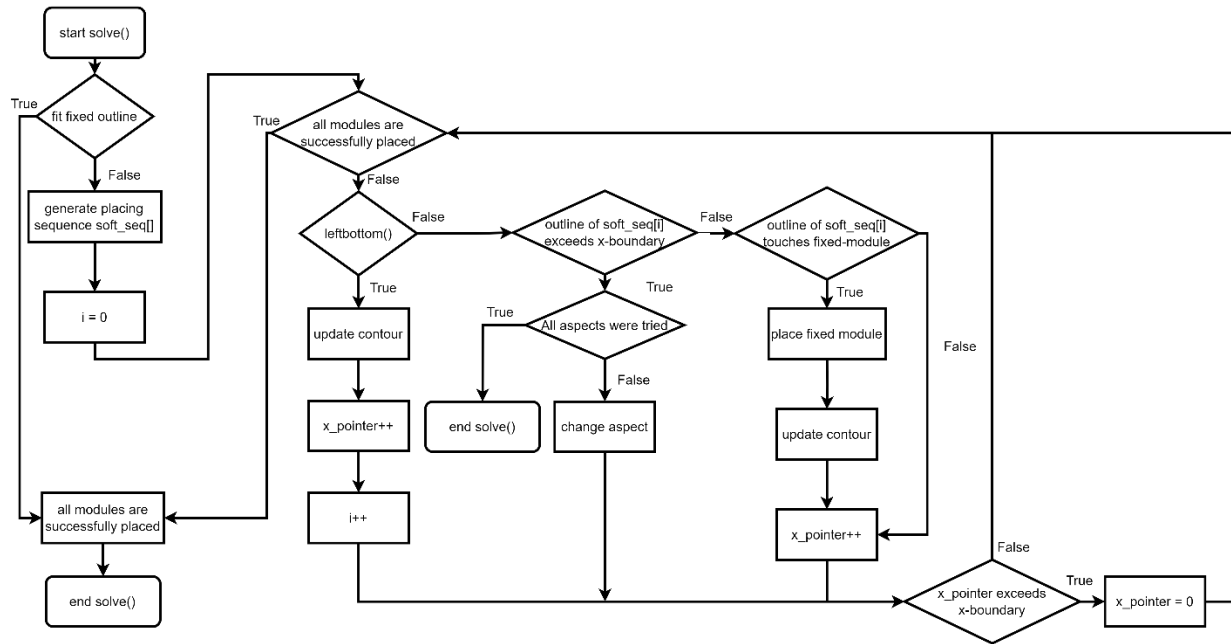
1. ParseInputFile  
Read the input file and store the information of fixed modules and soft modules.
2. PlaceFixedmodule  
Fill the location of fixed modules on the grid.
3. solve  
The details of this function will be illustrated by a flow chart in the following part.
4. CalculateHPWL  
Calculate HPWL by the current feasible solution.

$$\text{Total HPWL} = \sum \text{Manhattan Distance} \times \text{The number of connection (between two modules)}$$

5. File\_out  
Output the final result of this algorithm in the standard format including HPWL, the number of soft module and corner coordinates of each soft modules.

## (7) Important algorithms of our program

- solve()



**Fig. 4** Flow chart of function solve()

- leftbottom()

```

bool is_feasible = False;
copy_seq1 = copy_seq2 = soft_seq[i];
for k=1 to C do :
    if k != 1
        move outline of copy_seq1 left by 1 grid;
        move outline of copy_seq2 down by 1 grid;

        check if the outline in grids has enough space to put the module;

        if(large enough) then
            record this location of outline if it's better;
            is_feasible = True;
        else
            continue;

if(is_feasible = True) then
    put the module into Grid with the recorded location;
    return True;
else
    return False;
  
```

**Fig. 5** Pseudocode of leftbottom()

Every time we place a module, we will try to place it in the bottom-left position in order to make the modules closer. At the same time, we will also check whether the currently unacceptable move will be legal if we move it one unit to the left or bottom in the grid. This means that we can still overlap the outline of soft modules without violating their constraints.

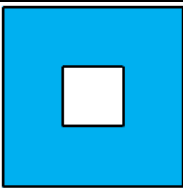
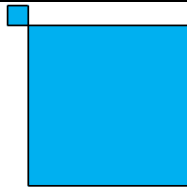
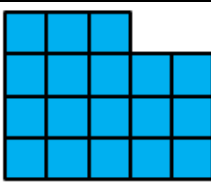

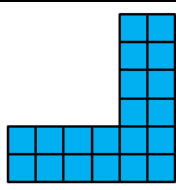
This function will return true and record the location of the if the left-bottom move is acceptable. It will return false if we can not find an acceptable placement. Note that the constant C in this pseudocode can be determined arbitrarily by user. If C is larger, it means that we can

try more possible bottom-left moves. However, it means that the time complexity will increase. If  $C$  is smaller, perhaps some solutions will not be considered but the time complexity will be reduced. Thus, there exists some trade-off for the choice of  $C$ .

- `check_constraint()`

After placing a module, we have to check if the placement of this module is legal. Note that type 1 will not happen because we will update contour and place the module in other location that is exclusive of the region of the fixed module. When selecting the aspects of the soft module, our algorithm automatically avoid shape of type 3, type 4, and type 5. Thus, we have to deal with type 2. The pseudo code and the example is shown below.

**Tab. 6** List of illegal module outlines

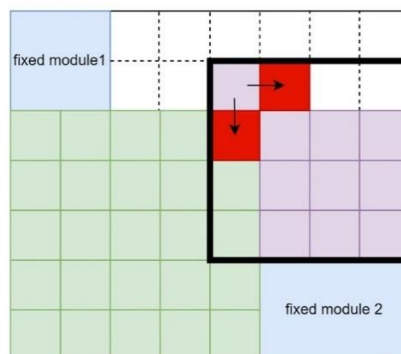
Illegal module outlines				
type 1	type 2	type 3	type 4	type 5
				
The region surrounded by the polygon has a hole in the middle.	The region surrounded by the polygon exceeds one.	The area of the soft module (18) is below the minimum area limit (20).	The aspect ratio violation (0.5-2).	The rectangular ratio violates the restriction (20%).

```

bool is_feasible = False;
for grid in outline of soft_seq[i] do :
    if (bottom grid != soft_seq[i] && right != soft_seq[i]) then
        return False;
return True;

```

**Fig. 6** Pseudocode of `check_constraint()`

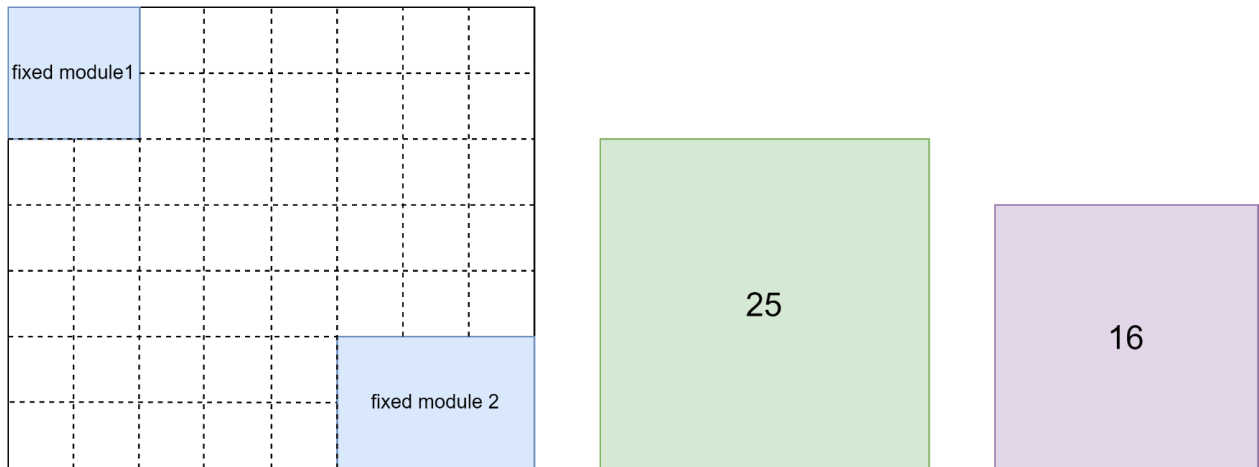


**Fig. 7** An example of `check_constraint()`

We have to check the left or bottom of the last filled grid has the same number as it does. If it does, it means that it is included in the same outline as the remaining grids that have the same number. If this situation does not hold, our algorithm will change the aspect of this module to retry again.

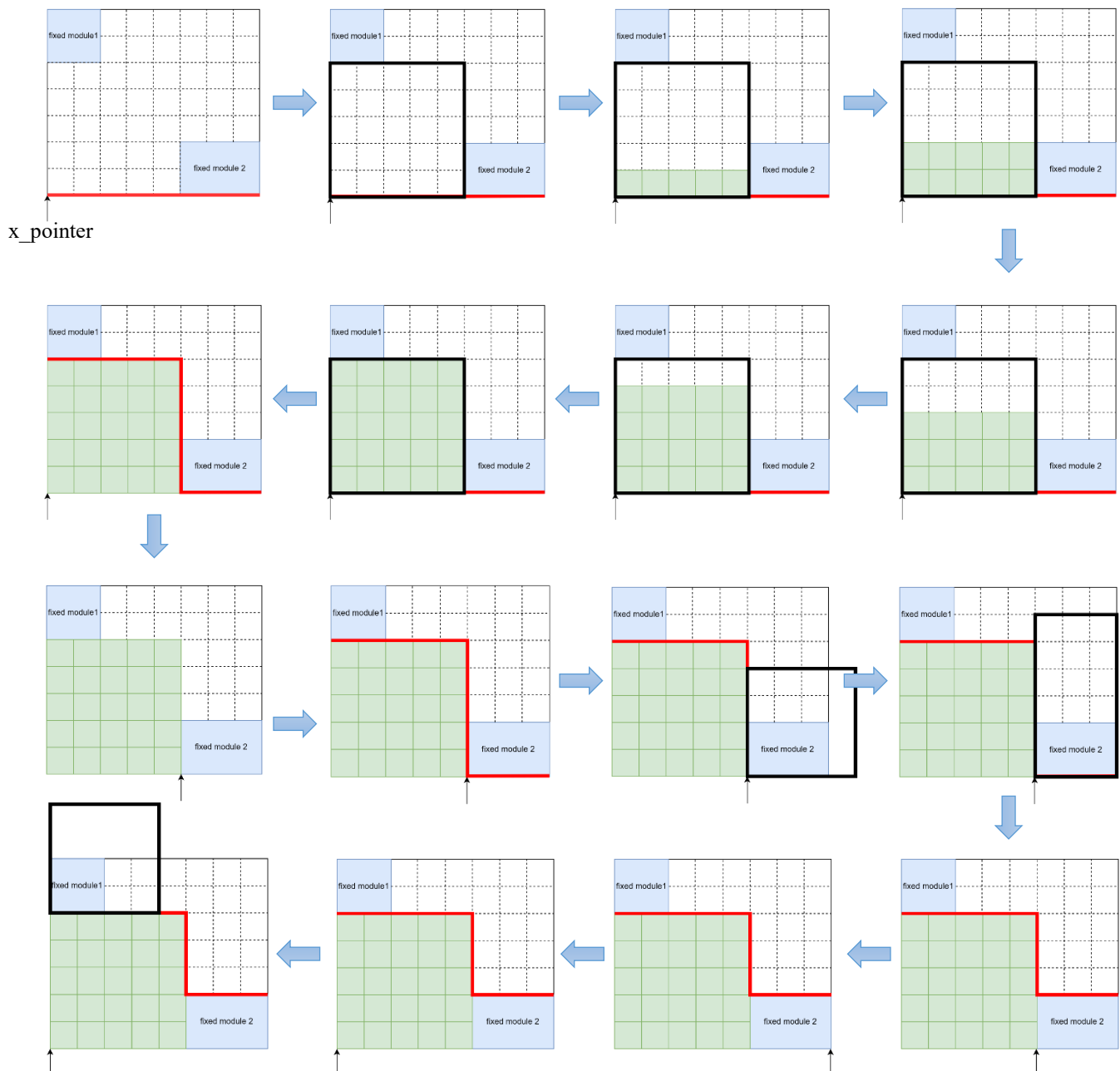


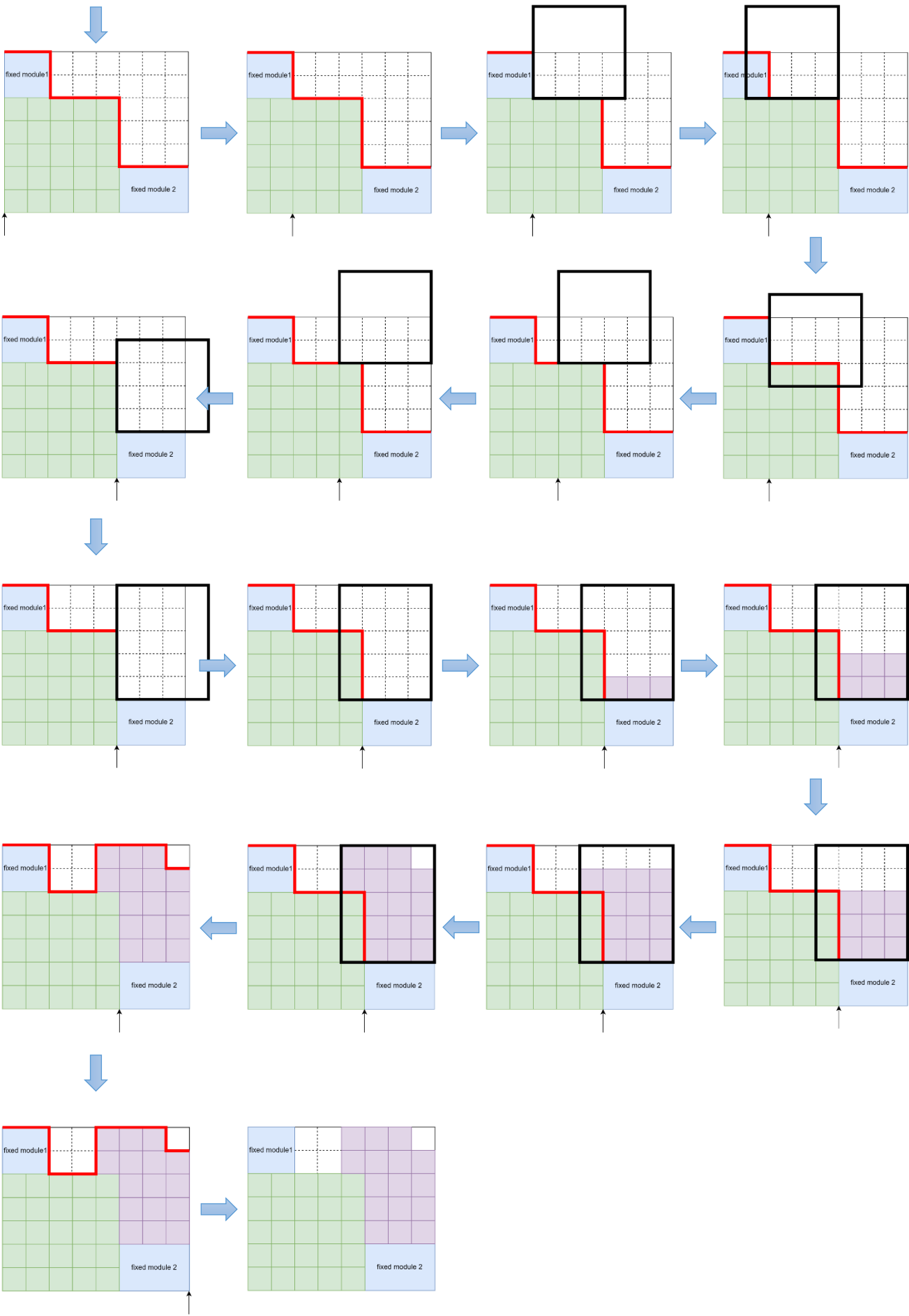
### (8) An example to illustrate our algorithm



**Fig. 8** The chip outline and modules being about to place in this example

The steps to place the green and purple modules:





## (9) How to execute our program

- Makefile

```

1 #CAD contest cpp compiler
2 .PHONY: all run clean
3 all: cadd0006.o
4 @g++ -std=c++11 cadd0006.o -o cadd0006_alpha.exe I.
5 run:
6 @./cadd0006_alpha.exe $(input) $(output) II.
7 clean:
8 @rm cadd0006.o
9 @rm cadd0006_alpha.exe
10 cadd0006.o: cadd0006.cpp
11 @g++ -std=c++11 -c cadd0006.cpp

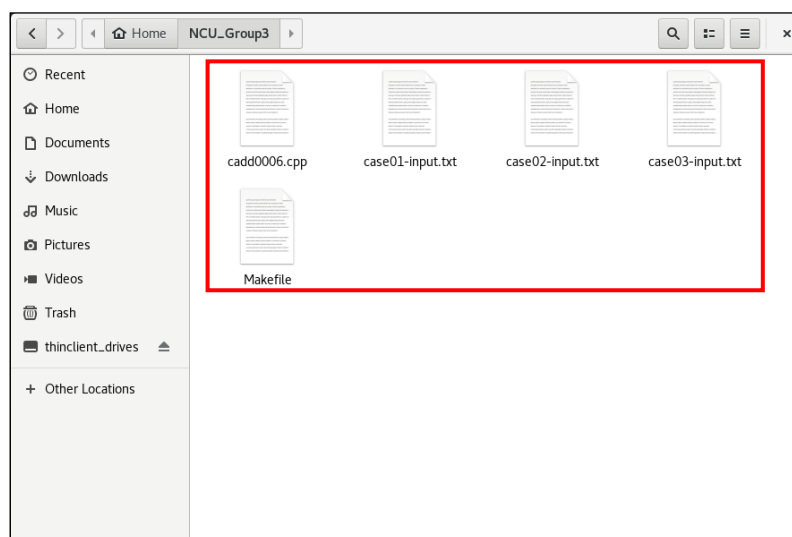
```

**Fig. 9** Code of Makefile

In order to make the compilation of our program simpler, we write a Makefile for the user to compile it. The steps to compile our program will be listed in the following parts. Here are some explanations of our Makefile:

- I. The required executable file name for alpha submission.
- II. The command to evaluate our program.

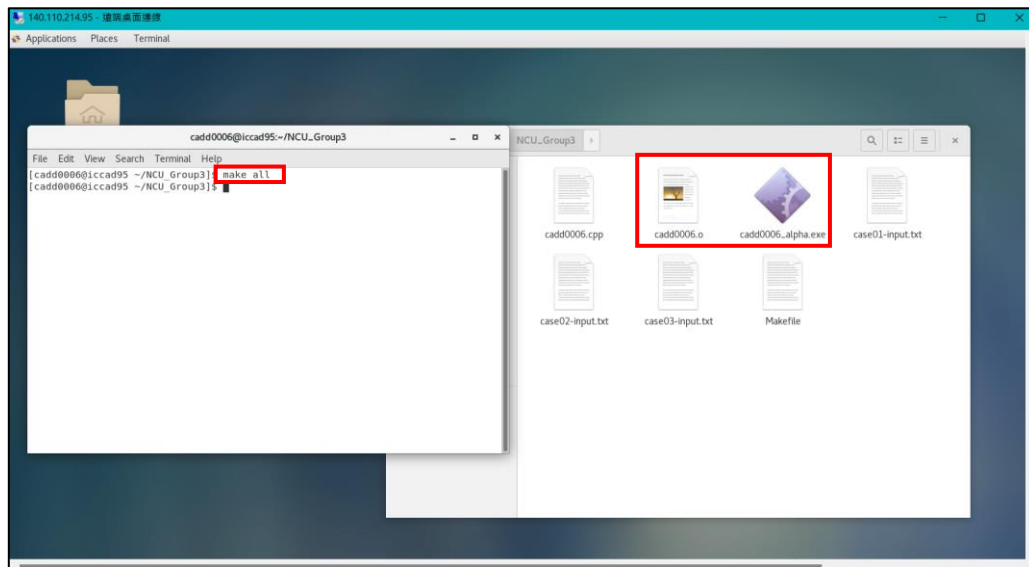
1. Make sure cadd0006.cpp, Makefile and all the testcases are involved in the same path.



**Fig. 10** Content of the folder for step 1

2. Type the command below to compile the source code and generate executable file.

`make all`



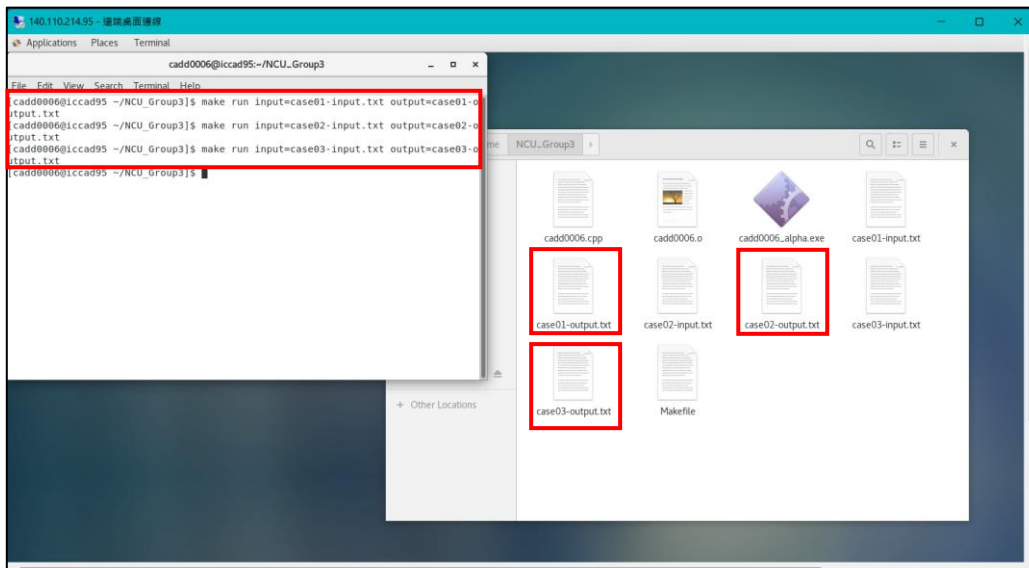
**Fig. 11** Files generated by make all

3. Type the command below to run execute the file. (Output files will be generated.)

`make run input=case01-input.txt output=case01-output.txt`

`make run input=case02-input.txt output=case02-output.txt`

`make run input=case03-input.txt output=case03-output.txt`



**Fig. 12** Files generated by make run

4. After finishing testing the code, we can use the command below to free the space in this folder. (Delete files generated by make all.)

make clean

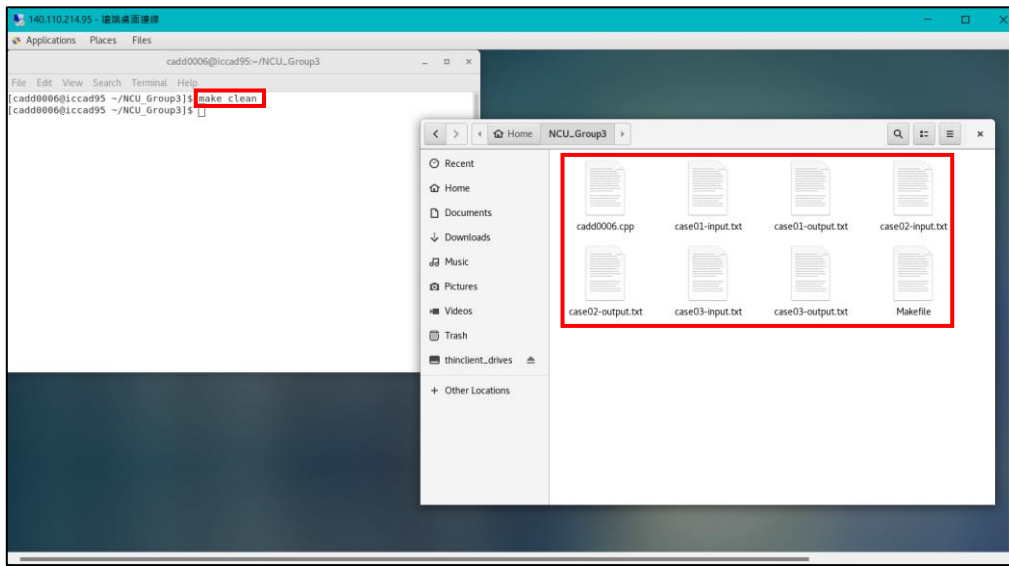


Fig. 13 Content of the folder after typing make clean

## (10) Results of our algorithm with public testcases

Tab. 7 Information of input testcases

Case	case01-input.txt	case02-input.txt	case03-input.txt
Chip width	11267	2300	2500
Chip height	10450	ss2300	3000
Number of soft modules	15	16	28
Number of fixed modules	5	8	14
Number of connections	45	40	108

Tab. 8 Results of each testcase

Case	case01-output.txt	case02-output.txt	case03-output.txt
HPWL	529800698	23803860	33756901

## (11) Conclusions

In our pursuit to address the various constraints posed by problem D, we have devised an innovative data structure and algorithm. This new approach goes beyond the limitations of the original version by effectively tackling both the fixed-module and overlapping challenges. What sets our method apart is the seamless integration of the contour concept within our algorithm, providing a straightforward means to verify the adherence of each module to the prescribed shape and location constraints.

**(12) Suggestions**

We believe that the reasons for our scores and deductions can be explained more clearly, specifically the meanings represented by each item. This way, we can make more targeted adjustments to our report.

Furthermore, we think that the documents gave us so clear explanations! The support materials are also very helpful! We really want to show our best appreciation to TAs and the professor! We enjoyed a lot and learned a lot from this project!! We are grateful for having this project, it helps us integrating data structure background into this modern floorplanning problems.

**(13) References**

- [1] ProblemD-20230328
- [2] 2023Spring\_EE6094\_CAD\_Final Project Checkpoint\_III\_20230511\_1132
- [3] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: enabling hierarchical design," IEEE Trans. on Very Large Scale Integration Systems, 11(6), pp. 1120–1135, December 2003
- [4] H. Murata and E. S. Kuh, "Sequence-pair based placement methods for hard/soft/pre-placed modules," in Proc. ISPD 1998, pp. 167–172.
- [5] L.-T. Wang, K.-T. Cheng, and Y.-W. Chang, Electronic Design Automation: Synthesis, Verification, and Testing, Elsevier/Morgan Kaufmann, 2009.
- [6] 2023Spring\_EE6094\_CAD\_Chapter9\_FloorPlanning\_20230413\_0300\_Stud