

## 一、 執行結果

```
[g114062529@ic22 HW3_grading]$ bash HW3_grading.sh
+-----+
| This script is used for PDA HW3 grading.
+-----+
host name: ic22
compiler version: g++ (GCC) 7.3.0

grading on 114062529:
  checking item | status
+-----+
  correct tar.gz | yes
  correct file structure | yes
  have README | yes
  have Makefile | yes
  correct make clean | yes
  correct make | yes

  testcase | hpwl | runtime | status
+-----+
  public1 | 82768800 | 7.39 | success
  public2 | 3619302602 | 52.51 | success
  public3 | 30814251320 | 169.88 | success
  public4 | 41712579740 | 160.21 | success
+-----+
| Successfully write grades to HW3_grade.csv |
+-----+
```

## 二、 LEF、DEF 檔補充說明

### 1. LEF:

- i. LAYER 段是對 routing layer 的設計規範，也就是 routing 在這層上接線時要遵守的規定。
- ii. VIA 段是對連接兩個 layer 的接孔的定義，會決定金屬層之間接線時要怎麼連結。
- iii. MACRO 段裡有很多 PIN，定義了一個 MACRO 上不同的 PIN 腳的位置以及其功能。

### 2. DEF:

- i. TRACKS 段規定了在晶片上哪些位置、以什麼間距、在哪一層金屬上可以畫線。
- ii. GCELLGRID 定義了 global routing grid 的座標，讓 global router 知道怎麼切成各個 grid 來進行 global routing。
- iii. SPECIALNETS 段定義的是電源與接地的佈線結構。

### 三、 Pseudocode

#### 1. FastDP:

```
Repeat
    Global swap
    Vertical swap
    Local re-ordering
N times
Single segment
```

#### 2. Global swap:

```
For each component C
    For each row R in C's optimal region
        Find each empty space S on R between optimal region
        if S is large enough and have benefit to WL
            put C to S and goto next component C
        Find each component T on R between optimal region
        if T is large enough and have benefit to WL
            swap C and T and goto next component C
```

#### 3. Vertical swap

```
For each component C
    For 2 rows R upper/lower to C's row
        Find each empty space S on R between optimal region
        if S is large enough and have benefit to WL
            put C to S and goto next component C
        Find each component T on R between optimal region
        if T is large enough and have benefit to WL
            swap C and T and goto next component C
```

#### 4. 其餘部分詳見下段說明

### 四、 程式碼詳細說明

- 實作 FastDP 演算法，主要架構與”An Efficient and Effective Detailed Placement Algorithm”論文中一樣，為 [ Global swap > Vertical swap > Local re-ordering ] 重複數次，最後再使用 Single segment 。

2. 沒有完全依照論文中的敘述重現各個子演算法，此外，做完 FastDP 後會再做幾次 Local re-ordering，可以讓結果更好。
3. Global swap:
  - i. 移動 component 時，對 optimal region 中的 row，先尋找 optimal region 中是否有空位可以放入，若沒有，再尋找 component 來交換。我認為直接移動 component 到空位較容易顯著減少線長，並且這樣操作也可以簡化實作難度，所以選擇這樣做。
  - ii. 交換過程中會保持 placement 是合法的，這點與論文中不同，原因是我採用論文中的方法時，可能是我估計線長的方法不佳，造成結果不佳，所以在移動 component 時使用以下這兩條規則。
  - iii. 論文中只要整條 row 尚有空間，都可以將 component 放上去並評估與其他 component 產生 overlap 的懲罰，但我難以準確估計該數值，因此在將 component C 放到其 optimal region 時，我「僅移動目標位置右邊的一個 component」，也就是要將 C 放到座標 tx 時，不能覆蓋到 tx 左邊的 component，僅可以嘗試將 tx 右方的 component R 往右移以空出足夠空間給 C，但若這樣會使 R 覆蓋到其他 component，則不能將 C 放上來。
  - iv. 將 component C 與其 optimal region 中的 T 交換時，若 C 周圍的空間小於 T 的大小，則不交換。也就是將 T 放到 C 的位置時，不調整其他 component 的位置，且不能有 overlap，否則不交換。
  - v. 也就是，當 C 與 optimal region 裡的 T 交換時，C 可以調整 T 右邊的 component 來騰出空位，但 T 不能調整其他 component。
4. Vertical swap:
  - i. 論文中僅寫到「嘗試將 component 往 optimal region 的方向移動一個 row」，因此我的做法是「尋找 component 往 optimal region 的方向(y 方向)的兩個 row，並且只尋找 row 上 x 座標在 optimal region 之間的區域」。只尋找 x 座標在 optimal region 之間的區域可以大幅減少計算時間，並且搜尋兩個 row 以增加成功的機率。
  - ii. 其餘與 Global swap 相同，他們的差異只有搜尋的 row 不同。
5. Local re-ordering
  - i. 相鄰的三個 component 為一組，嘗試六種排序，找最好的那種。
  - ii. 實作上排序時，左界為 component 1 的左邊，右界為 component 3 的右邊，component 2 則與 component 1 相鄰擺放。

## 6. Single segment

- i. 沒有採用論文中的 clustering 方法，僅應用了論文中提到的「線長函數會是一個凸函數」的特性。
  - ii. 對每個 component 在 row 上能移動的範圍中，選最左、最右、中間三點，計算一個二次方程式，找出這段空間中線長最小的點。
7. 其餘在實作上主要的細節有以下幾點：
- i. 在每個 row 上有 `vector<Component*> comps`; 會紀錄所有在該 row 上的 component，並依照 x 座標的大小由小到大排序，搭配 STL 的 `lower_bound()` 函式來快速完成搜尋操作。
  - ii. 每個 component 有 `int sx1, sx2`; 目的是將原始 x 座標轉換為對齊 site 位置的座標。
  - iii. 每個 net 上有 `vector<Component*> compsX` 跟 `compsY`，將 net 上的 component 事先分別依照 x 跟 y 座標排序，要計算線長時就可以直接找到最大跟最小的座標。

## 五、學到什麼/遇到的問題

這次遇到最大的困難是對於一個 component，如何尋找在 row 上與他相鄰的另一個 component。原先使用的方法太過複雜，使程式碼長度暴增、很容易出現 segmentation fault、以及我很難 debug。仔細思考過後改為使用 `vector<Component*>` 去紀錄每個 row 上的 component，將其排序好，就能使用內建的 `lower_bound()` 函式來快速搜尋、移除、插入。這點讓我學到在寫這種較為大型的程式碼時，應該要先將整體架構想好，先思考具體會需要什麼操作，判斷用什麼資料結構會比較方便，再開始寫程式碼，才能達到事半功倍的效果。

## 六、AI 工具使用紀錄

1. 使用 `chatgpt` 來協助我完成這項作業，內容如下。
2. 協助我了解讀檔、寫檔部分，包含 `istringstream`、`ifstream`、`ofstream` 的使用方式。
3. 協助我了解 `lower_bound()` 函式以及 `vector<T>::iterator` 的使用方式。
4. 協助我了解 `nth_element()` 函式的使用方式。
5. 協助我完成我的 `solveEq()` 中尋找二次方程式的部分。