

# CS 512200 VLSI Design for Manufacturability

## Final Project: Dummy Fill Insertion

110062802 呂宸漢

### I. Algorithm

由於本次 final project 的目標是要在滿足所有 constraint 的情況下最小化因插入 filler (dummy fill) 而與 critical net 產生的 lateral capacitance。而 constraint 又分為三類，分別是 min spacing constraint、min/max fill width constraint 與 min/max metal density constraint。其中 min spacing constraint 只需在找可以插入的空間時將 conductor 的四個邊都往外擴 min spacing 的距離，即可確保在空間內插入的 filler 都滿足 min spacing；而 min/max fill width constraint 則是在找到可以插入的空間之後依照 width 的限制將 filler 切小塊或是過濾掉不合法的 filler 即可；比較麻煩的是要如何插入與刪除 filler 使每個 window 的 density 滿足 min/max metal density constraint。因此，我設計了一個演算法依照滿足 constraint 的難易度分成五個步驟解決 dummy fill insertion 的問題。

#### A. Overview

**Figure 1** 是我的演算法的流程圖。由於每層 layer 可以獨立安插 filler，因此我只需實作一個安插演算法並套用在每一層 layer 上即可，我的安插演算法主要分成以下五個步驟進行。

##### 1. Initial Tile Grid

在讀取完 input file 後我會將 design 切成一個一個 tile，每個 tile 的大小與 window 的 step size 相同，tile 內會記錄有哪些 conductor 與哪些 filler 會占用這個 tile 與他們的占用面積(occupy area)，如此一來在計算 metal density 時，只需要將 window 有包含到的 tile 的占用面積加總並除以 window area 即可，可以節省一些重複計算的時間。不過在計算 conductor 占用的面積時需要注意同一條 net 的 conductor 是否有互相重疊，如果有重疊的話則需要扣掉重疊的面積以免高估，因此我實作了一個排容演算法解決此問題，我會在 Section B 介紹。

##### 2. Generate Candidate Regions

在初始化 tile grid 後，我會先一個一個 tile 找可以合法插入 filler 的區域，並檢查如果這些區域全部都插入 filler 後能不能滿足 min metal density constraint，如果可以就進到下個步驟；如果不行，則代表一個一個 tile 各自尋找區域會浪費太多可用空間，因此我會改成從整個 chip 尋找可用空間，以解決有可用空間卻被 tile 切開的問題。此部分所使用的演算法我會在 Section C 介紹。

### 3. Insert all Fillers

找到所有 candidate region 後，我會將每個 region 轉成 filler 插入，由於有些 region 的大小可能因為 min/max fill width constraint 的限制而無法直接轉成 filler，因此我會先過濾掉小於 min fill width constraint 的 region，並將太大的 region 用 max fill width 計算可以切成幾塊大塊的 filler，再將這些 filler 全部插入以滿足 min density constraint。

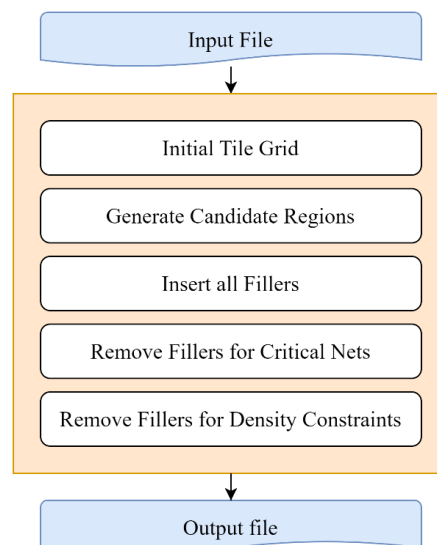
### 4. Remove Fillers for Critical Nets

由於本次作業是要比較在 critical net 上產生的 lateral capacitance 的大小，因此我會在確定可以滿足 min metal density constraint 的情況下優先移除或縮小 critical net 旁的 filler 以將低 lateral capacitance，為了快速取得影響較大的 filler，我會將 critical net 的 conductor 的 boundary 放大兩個 min spacing，蒐集有與其重疊的 filler 並記錄其 lateral capacitance，最後再依照 lateral capacitance 由大到小依序檢查移除後是否可以滿足 min metal density 再移除 filler。

### 5. Remove Fillers for Density Constraints

在移除完所有與 critical net 有關且可以移除的 filler 後，我會再移除多餘的 filler 以滿足 max fill width constraint，移除方式則是一個一個 tile 檢查是否可以移除 filler，若移除後仍滿足 min metal density constraint 就移除。

當每一層 layer 都執行完上述五個步驟後，就可以得到每個 layer 需要插入的 filler 的座標，再將這些資訊寫成 output 輸出即完成整個 dummy fill insertion 的 process。



**Figure 1:** Flow chart.

## B. Conductor Occupy Area

**Algorithm 1** 為計算 tile 中 conductor 占用面積的演算法。首先，我會先加總所有 conductor 有與 tile 重疊的面積，然後再利用排容原理修正多個 conductor 重疊時多算的面積。由於只有同一條 net 的 conductor 會重疊，因此在做排容時只需檢查同一條 net 的 conductor 即可，為了加速計算，我會先將 conductor 編號並且在檢查重疊時只跟編號比他大的 conductor 檢查，這樣就可以減少許多重複的檢查，並且依照若有  $n$  個 conductor 重疊必先有  $n - 1$  個 conductor 重疊的性質，可以只檢查已有重疊的 region 與編號比他大的 conductor 是否重疊，進一步縮小 solution space 並且依照排容原理加上或減去重疊的面積即可在短時間內正確計算一個 tile 內被 conductor 占用的面積。

```

Input: tile boundary  $b$ , conductor list  $C$ , net list  $N$ 
Output: occupy area  $\alpha$ 
 $\alpha \leftarrow$  add up the intersecting area of  $b$  and  $c$ ,  $\forall c \in C$ ;
foreach net  $n$  in  $N$  do
    build a list  $CL$  to record  $c$  and the index number  $i_c$  of  $c$  in  $CL$ ,  $\forall c \in n$ ;
    build a list  $RL$  to record the intersect region of  $b$  and  $c$  and  $i_c$ ,  $\forall c \in n$ ;
     $s \leftarrow -1$ ;
    while  $RL$  not empty do
        initialize an empty list  $IL$  to record intersecting region and index number;
        foreach region  $r$  and index  $i$  in  $RL$  do
            for  $j \leftarrow i + 1$  to  $\text{size}(CL)$  do
                if  $r$  and  $CL[j]$  intersect then
                    add intersect region and  $j$  into  $IL$ ;
                     $\alpha \leftarrow \alpha + s \times$  intersect area;
                endif
            endfor
        endfor
         $s \leftarrow -s$ ;
         $RL \leftarrow IL$ ;
    endwhile
endfor
return  $\alpha$ ;

```

**Algorithm 1:** Calculate conductor occupy area.

### C. Generate Candidate Regions

由於 filler 與 filler 間和 filler 與 conductor 間都需要滿足 min spacing constraint，因此我們需要找一個快速且有效的方法確保可以找出最大量的可用空間並且在轉換成 filler 時可以浪費最少的空間。為了最佳化所有空間運用，我在尋找可用空間時會先將所有 conductor 的四個邊都往外擴張半個 min spacing，並利用掃描線演算法找出所有可用空間。如此一來，在將可用空間轉換成 filler 時，無需考慮空間與空間是否有間隔 min spacing 的距離，只需先過濾掉長或寬小於 min spacing 與 min fill width 的總和的可用空間，並將長或寬大於 max fill width 的可用空間切成多塊可以滿足 max fill width 的最大空間，最後將這些可用空間的四個邊都往內縮半個 min spacing 就可以轉換成 filler，且 filler 間必定保持一個 min spacing。

雖然掃描線演算法可以快速且有效地找到所有可用空間，不過因為題目的關係，有些可用空間會因為長寬小於 min fill width 而無法使用導致空間浪費，因此我實作了一個兩階段的演算法解決此問題。

#### 1. Get All Free Regions

**Algorithm 2** 是基於掃描線演算法實作的尋找所有可用空間的演算法。由於本作業有 min fill width 的限制，掃描的品質與 conductor 的方向有正相關，因此我會先判斷每個 layer 的 routing direction 再決定掃描的方向。為了得到較高品質的結果，我使用垂直的掃描線去掃水平的 layer，若遇到垂直的 layer 時我則將所有座標的 x 與 y 互換，在得到所有可用空間後再將座標換回來。正如先前所述，我在建立掃描線的時候會先將所有 conductor 的四個邊都往外擴張半個 min spacing，再紀錄每個 conductor 的左右邊界。建立完掃描線後我就會一條一條掃描線看目前有哪些 conductor 被掃描線切到並記錄在 cut set，然後再從 cut set 中尋找空白的垂直區間並記錄在 free interval set。由於有 min fill width 的限制，我們會希望盡量找到大的且連續的區間以方便插入 filler，因此我會先從 temp region list 中檢查每個 temp region 是否可以再繼續延伸，此處我是直接判斷 temp region 的上下邊界是否有在 free interval set 中。如果有則繼續延伸 temp region 並將此 interval 從 free interval set；如果沒有則代表此 temp region 對多只能延伸到這，因此我會將 temp region 的右邊界設成當前掃描線的 x 並將 temp region 從 temp region list 移到 region list。當所有 temp region 都檢查完畢後，便可以將剩餘的 free interval 都轉成 temp region，temp region 的左右邊界設成當前掃描線的 x，而上下邊界就設成 interval 的邊界。待掃描線全部掃完後，即可得到所有的可用空間。

```

Input: tile boundary  $b$ , conductor list  $C$ 
Output: free region list  $RL$ 
initialize two empty region lists  $RL$  and  $RL'$ ;
build a sweep line set  $LS$  to record the expanding boundary of  $c$ ,  $\forall c \in C$ ;
foreach sweep line  $l$  in  $LS$  from left to right do // in x-axis
    build a cut set  $CS$  to record cut conductors by  $l$ ;
    find all free interval  $i$  in  $CS$  and store them in free interval set  $IS$ ; // in y-axis
    foreach temp region  $r'$  in  $RL'$  do
        if interval( $r'$ ) is in  $IS$  then
            remove interval( $r'$ ) from  $IS$ ;
        else
            update the right edge of  $r'$  to  $l$ ;
            move  $r'$  from  $RL'$  to  $RL$ ;
        endif
    endfor
    generate temp region  $r'$  according to free interval  $i$  in  $IS$  and add into  $RL'$ ;
endfor
return  $RL$ ;

```

**Algorithm 2:** Get all free regions.

## 2. Refine Free Regions

由於在第一階段只是找可用的空間，可是不代表每個空間都可以容納 filler 的，尤其又有 min fill width 的限制，因此最簡單的解決方式就是直接將不合法的可用空間去除，只留下合法的可用空間去插入 filler。此方法雖然簡單暴力，但是有些不合法的空間或許可以與旁邊的合法空間合併，就可以得到更大的可用空間。為此，我又再寫了一個以掃描線演算法為基礎的優化空間演算法 **Algorithm 3**，與先前的掃描線演算法一樣，這個演算法也依據 routing direction 作同樣的修正。為了所小 solution space 並簡化問題，我只有考慮三個 case 並在達成特殊條件時才會合併。以下 case 所討論的對象都是前者的右邊界與後者的左邊界在同一條掃描線上的情形。

case 1: 前者合法且後者合法

判斷兩者的上下邊界是否相同，若相同就將兩者合併成一個塊。

case 2: 前者合法且後者非法

判斷後者的上下邊界是否可以包含前者，若可以則前者向後延伸。

case 3: 前者非法且後者合法

判斷前者的上下邊界是否可以包含後者，若可以則後者向前延伸。

由於這個部分有許多較困難的時作細節，因此我就簡要帶過，如果對如何實作有興趣可以自行參閱「DensityMamager.cpp」內的「refineFreeRegion」。

```

Input: free region list  $RL$ 
Output: refined region list  $RRL$ 
build a sweep line set  $LS$  to record the boundary of free region  $r$ ,  $\forall r \in RL$ ;
foreach sweep line  $l$  in  $LS$  from left to right do // in x-axis
    foreach region  $rr$  which has the right boundary in  $l$  do
        foreach region  $lr$  which has the left boundary in  $l$  do
            if  $rr$  is legal and  $lr$  is legal then // case 1
                if  $rr$  and  $lr$  have the same top and bottom boundary then
                    set the right boundary of  $rr$  to the right boundary of  $lr$ ;
                    remove  $lr$  from  $LS$  and update  $rr$  in  $LS$ ;
                endif
            else if  $rr$  is legal and  $lr$  is illegal then // case 2
                if the top and bottom boundary of  $lr$  can cover  $rr$  then
                    set the right boundary of  $rr$  to the right boundary of  $lr$ ;
                    update  $rr$  in  $LS$ ;
                endif
            else if  $rr$  is illegal and  $lr$  is legal then // case 3
                if the top and bottom boundary of  $rr$  can cover  $lr$  then
                    set the left boundary of  $lr$  to the left boundary of  $rr$ ;
                    update  $lr$  in  $LS$ ;
                endif
            endif
        endif
    endfor
endfor
foreach region  $l$  in  $LS$  do
     $RRL \leftarrow$  store the region which has the left boundary in  $l$ ,  $\forall l \in LS$ 
endfor
return  $RRL$ ;

```

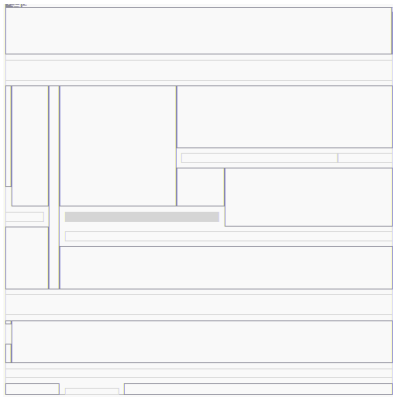
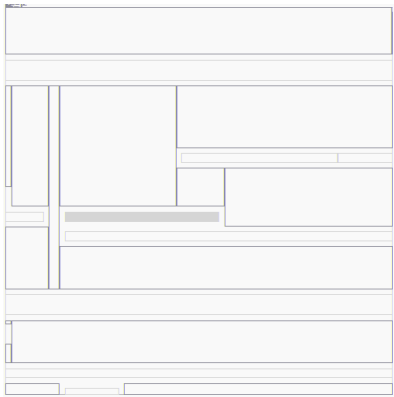
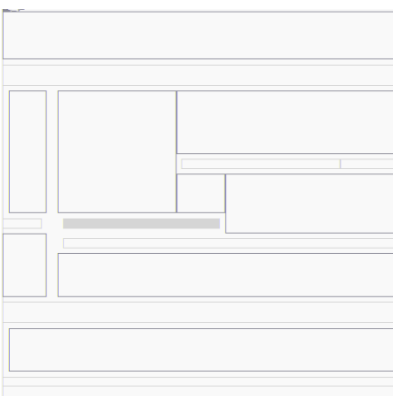
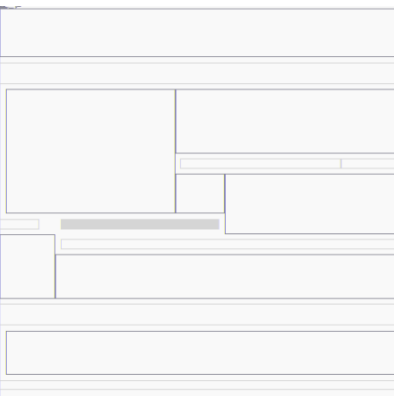
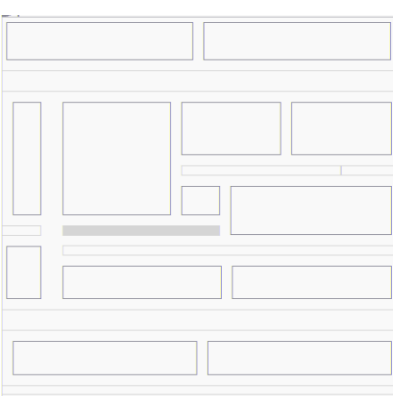
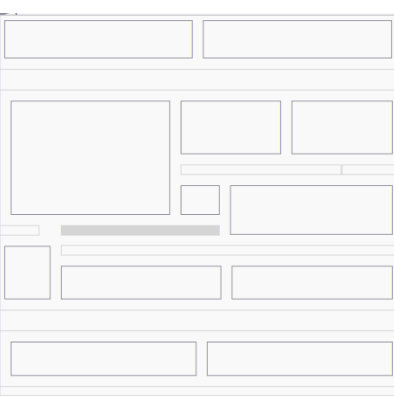
**Algorithm 3:** Refine free regions.

#### D. Visualize Tile Layout

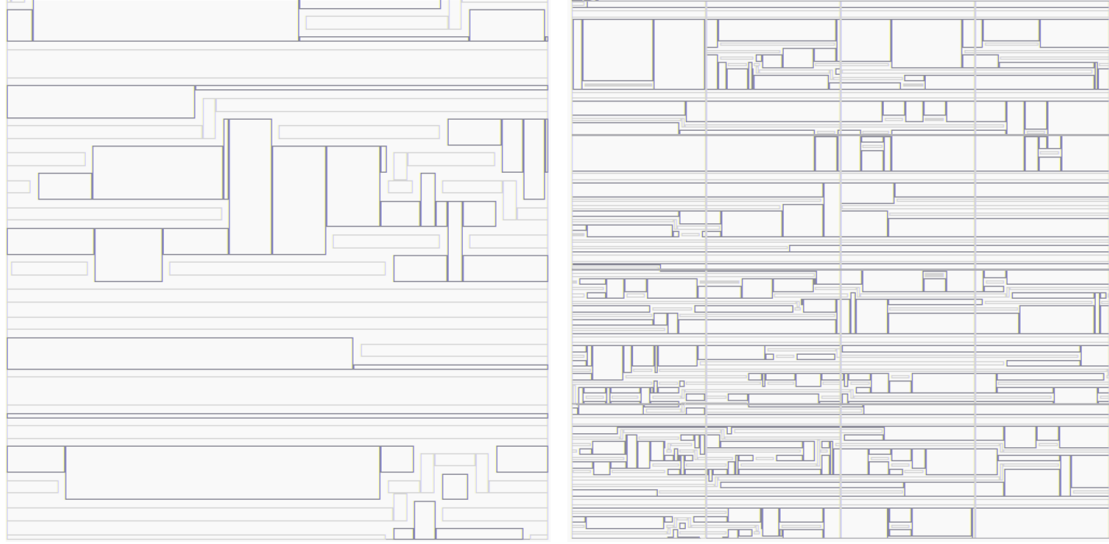
Table 1 中每一列圖片的粗線分別代表一個 tile 的可用空間(free regions)、可以插入 filler 的空間(legal regions)和插入 filler 後的結果(fillers)，而圖片中較細的線為 conductor。左邊的圖片為用 Algorithm 2 找到可用空間後直接過濾非法空間再插入 filler 的結果，右邊的圖片為用 Algorithm 2

找到可用空間後用 **Algorithm 3** 合併非法空間再插入 filler 的結果。由 **Table 1** 可以看出 **Algorithm 2** 可以框出所有的可用空間，不過仍然會有一些非法的空間。若是直接過濾掉非法空間，可以插入的 filler 面積就變少，可能會導致該 window 無法滿足 min density constraint。若改用 **Algorithm 3** 則可以合併更多非法空間讓可以插入的 filler 的面積更大。

**Table 1:** Comparison between Filter Illegal Regions and Refine Free Regions

Figure Type	Filter Illegal Regions	Refine Free Regions
Free Regions		
Legal Regions		
Fillers		

為了檢視 **Algorithm 2** 的效用，我也有再印了一張比較壅擠的 tile 與 window 的圖出來看，由 **Figure 2** 可以看出 **Algorithm 2** 的確可以找到所有可用空間，只是不保證所有空間都合法。



**Figure 2:** The visualization of a tile (left figure) and a window (right figure).

## II. Experimental Results

由於我的方法是一開始插滿所有 filler，然後再依照 filler 影響 critical net 的嚴重程度依序移除 filler，以最小化 lateral capacitance。因此，在實驗結果的部分我會比較完全插滿 filler 的方法(fill all fillers)與插滿後先移除會影響 critical net 的 filler 再移除多餘的 filler 的方法(reduce fillers)兩種方法的 weighted capacitance 與 runtime。由 **Table 2** 可以看出兩個方法的 runtime 都很快，雖然 reduce fillers 在 weighted capacitance 的方面大勝 fill all fillers，可是 runtime 就會輸一點點，主要是因為每次移除都要檢查 density 的緣故，不過這個 runtime overhead 還算是可以接受的範圍。

**Table 2:** Quality comparison between Fill All Fillers and Reduce Fillers.

Testcase	Weighted Capacitance		Runtime (s)	
	Fill all fillers	Reduce fillers	Fill all fillers	Reduce fillers
3	1,306,966	313,517	0.48	0.63
4	2,664,383	641,690	1.07	1.37
5	5,305,816	1,279,582	2.10	3.47