Flip-Chip Routing

第九組

模型

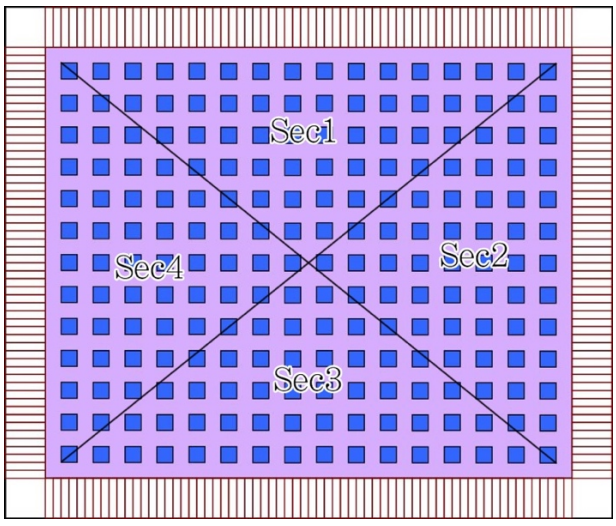
我們開了一個矩陣 graph，裡面存放 class Driver，Driver 裡包含 bump 的位置、座標、index、所在區域、目的區域等等資訊。

利用 global routing 排出的路線座標點也會存在此矩陣中，最後再藉由 detailed routing 接線並座標化、輸出線段。

演算法

1. 寫入及初始化（馬靖孟）
   I. 每個 bump 有自己的 index、x0、xf、y0、yf。
   II. 對每個 bump 初始其 virtual_index = index。
   virtual index 是為了 cross section bump 所新增的判斷條件。

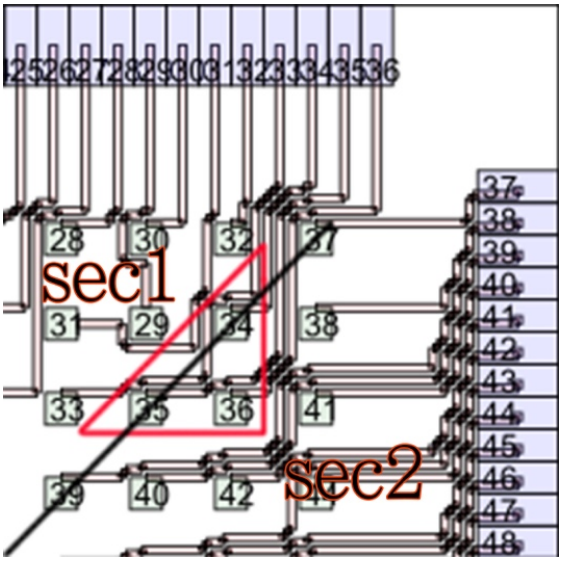
2. Partition（陳立馨）
   步驟
   I. 以 bump 的頂點到整張圖的中心點連線的斜率作為依據，利用對角線將所有 bump 劃分成四個區域。



   II. 紀錄 bump 自己在的區域.sec[0]及對應 driver 在的區域.sec[1]。將 bump 和 driver 在不同區的 bump 設為 cross-section bump。如果

bump 剛好在對角線上，我們會根據 bump 的頂點與中心點連線的鞋綠，和對應 driver 所在 section 去判斷他要被分在哪個 section。

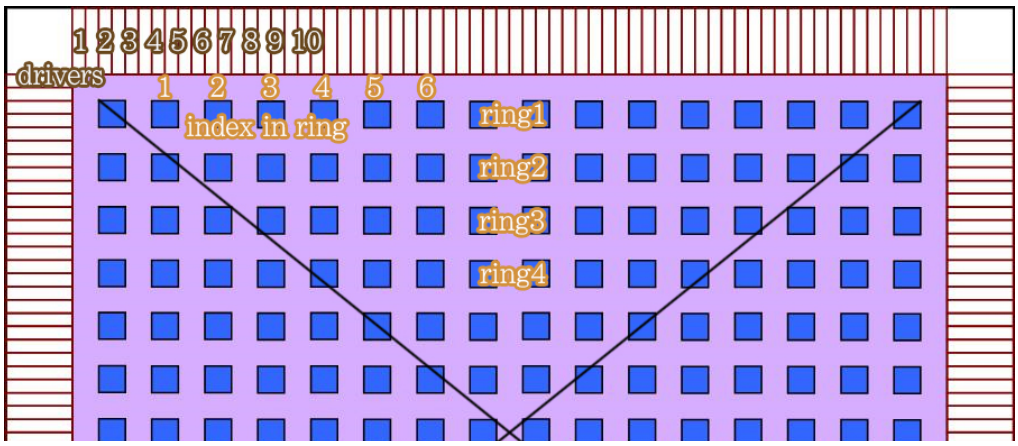
III. 另外有一種特殊情況是針對 cross section bump 做判斷，當 cross section bump 鄰近我們所劃分的分界的時候，他會判斷自己有沒有機會被擺在正確的 section，而不用當 cross section bump。像是 test144 中的 bump36，他原本會被分在 sec2 但是因為他鄰近邊界，且左邊（bump35）以及上面（bump34）都是 sec1，所以他可被重新劃分到 sec1。因為我們是假設 cross section bump 沒有很多的情況，所以我們沒有再進一步做 recursion。



bump36 可藉由判斷左方及上方的 bump，來讓自己變成非 cross-section

3. Set Ring（陳立馨）

甲、將每個 section 離 driver 最近的一層 bump 定他的 ring=1，每多一層則 ring++。並且設置 index_in_ring，即 bump 在該層 ring 的順序跟著 driver 小至大的順序排。



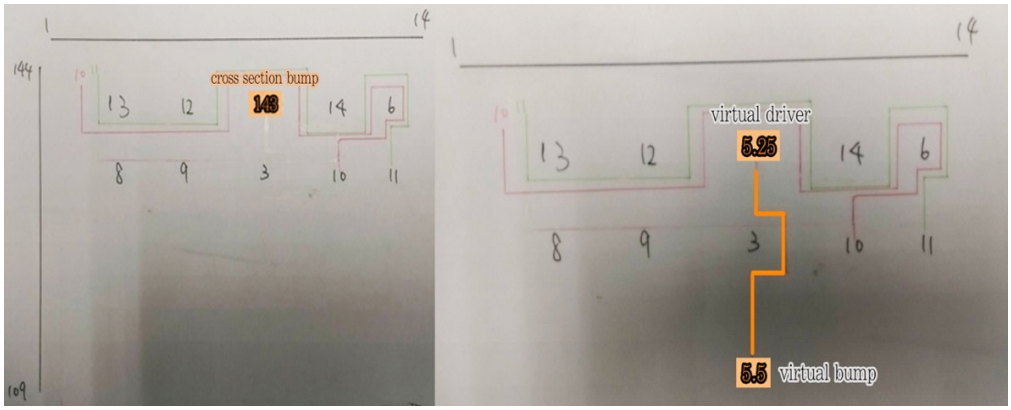
限制：因為題目有限制說 driver 本身是照著大小順序排好（逆或順時針），故因此設計。

4. Set Virtual Bump（陳立馨、陳溥生）

此概念為將原本的 bump 視為新的 driver，virtual bump 視為 bump 去做 global_routing。

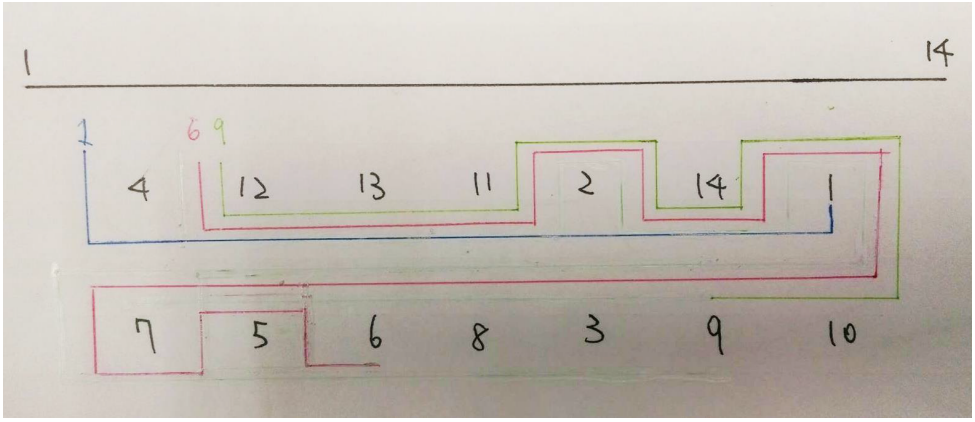
I. Cross section bump 在其終點區域加開一層 ring，並設置該區域的 virtual bump。

II. Cross section bump 在自己的區域，也加開一層 ring，並於該層設置 virtual bump。譬如：原有 5 層 ring，則在第 6 層 ring 增加 virtual bump。



III. 由於 global routing 的判斷方式，在此 case 中我們只需要將原本的 cross section 的 index 設為同層（ring）current_index 比他大的數字中最小的 index-0.5，我們將此 index 稱為 virtual index，在做判斷繞線的時候都是在比 virtual index。

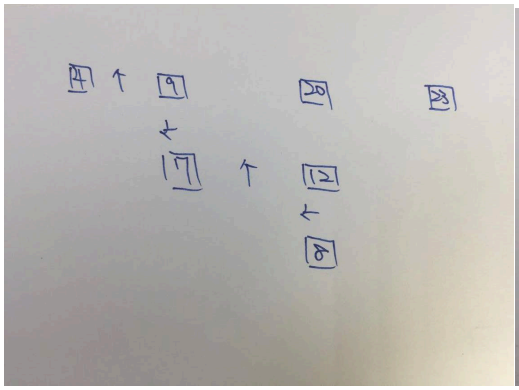
5. Global Routing（陳溥生）



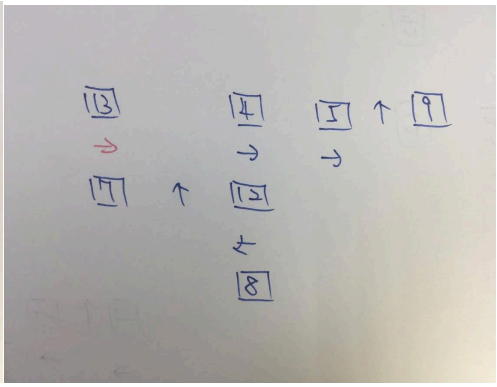
I. 由於我們將 bump 先排順序之後存進 vector，在這個 case 中我們是由左至右、由上至下的存，而處理 global routing 的時候我們先從同層

（ring）離 driver 最近一層開始做起，在這個 case 中因為 driver 從小到大是由左至右，所以我們會先從左邊的 bump 開始接線。而從第二個開始會進行繞線，繞線的方式是與同層 index_in_ring 較小（該 bump 左側）的 bump 比較，比他大的走下面，比他小的走上面。

進行完同層之後我們會移到下一層，會從下一層的最後一個 bump 開始比較，當比到比自己 virtual_index 小的時候我們就會跳出迴圈，而在此 bump 右邊設置一個向上的箭頭。同時我們會比較他們的相對位置補上對應的箭頭。需注意的是在第一次進入下一層的時候在自己的旁邊要補上箭頭，但第二次進入的時候不用。



圖片 a



圖片 b

II. 根據上圖 a，8 在第一次進行到下一層得時候會先搜尋比他小的數字，找到 7 之後會在 7 的右邊放一個向上的箭頭，並因為相對位置（7 相對 8）在比較的位置上方（目前是 8）補上向左邊的箭頭，然後將下次要比較的位置定在 7。而在進入第二層的時候同理會找到 4（與 8 相比），並在 4 的旁邊放向上的箭頭，再因為相對位置（4 相對 7）在比較的位置上方（目前是 7）擺上向左的箭頭。

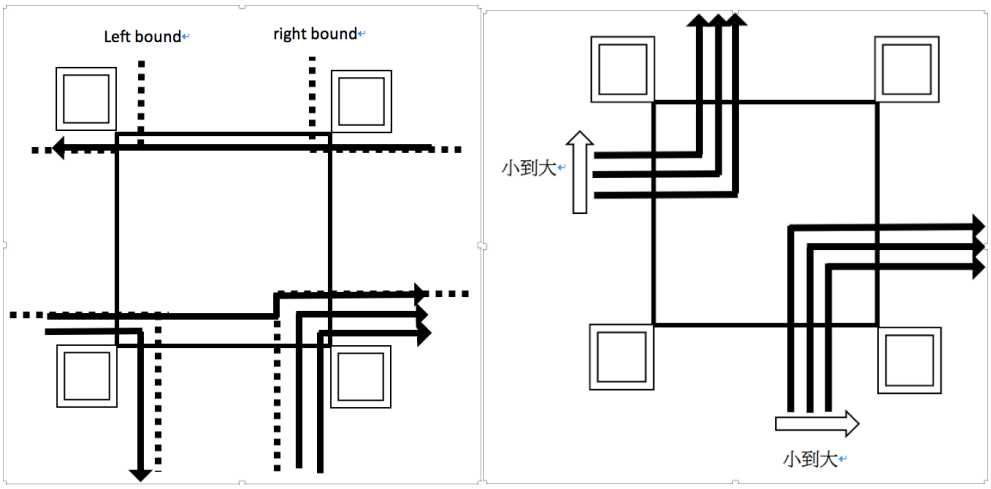
但如果看到圖 b，在第二次進入下一層的時候會先找到 5 並在 5 的左邊補上箭頭，再因為現在比較的位置是 7（7 相對於 5）而想要補上紅色的箭頭，但我們並不需要紅色的箭頭，故需要判斷目前是否為第一次進入下一層來判斷要不要補上相對位置上方的箭頭。

III. 而在使用 virtual bump 的時候因為是把 bump 視為 driver 所以我們要多判斷如果兩 virtual bump 相減值小於一的話，這次繞線就會停止。

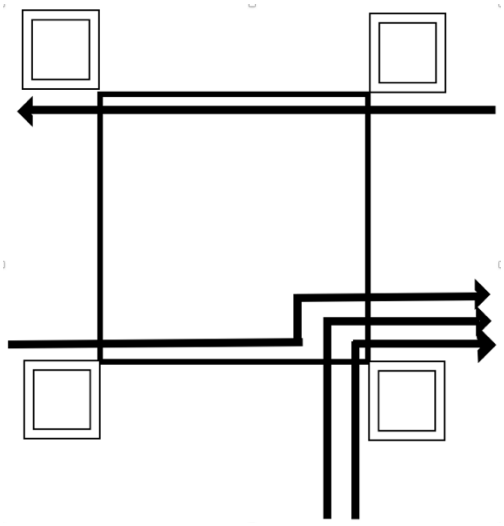
缺點：由於我們沒有辦法判斷是否該路徑已經爆量，在這個 case 因為兩 bump 之間只能塞四條線，而我們有可能會超過四條線，所以我們這個 global routing 的限制是他的 bump 排序需要盡量的照順序。但我們這個繞法一定能繞出路徑且不會打架，而且方便好懂。

6. Detailed Routing（馬靖孟、葉津源）

I. 經過 Global routing 之後，我們能夠知道每兩個相鄰的 bump 之間有哪條線經過，以及那條線是往哪個方向走。我們把四個 bump 圍成的區域如上圖稱之為 tile。經過每個 tile 的路徑我們把它分為 12 類，分別是左邊進上面出、左邊進右邊出、左邊進下面出以此類推，共 12 種路徑。由於 driver 的編號大小是順時針由小到大，我們發現通過同樣路徑的線之間是會按照一定大小順序排列的，例如所有逆時針轉彎的線，靠近 bump 的線 index 會較小，然後由小到大向外排序，反之順時針的線則是從靠近 bump 的線由小到大向外排。直線一律靠近前進方向的右手邊，並從前進方向右手邊往左由小到大排列。同時，當直線遇到轉彎時，轉彎要比直線先占靠近 bump 的位置。



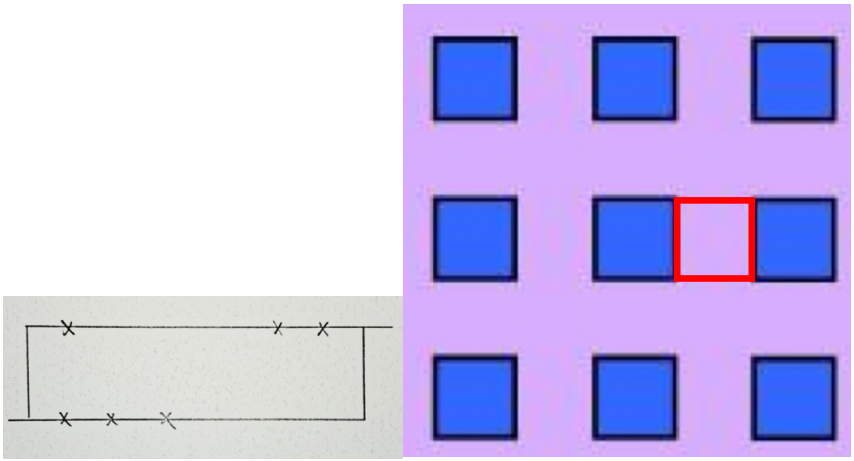
II. 在做 detail routing 時，我們針對每個 tile 的四個邊分別記錄兩個邊界，接下來從轉彎的線開始記錄，每紀錄一條路線，相對應的邊界就往內移一段距離，距離視線寬與線間距而定。跑完八種轉彎後，接著處理四種直線。直線會根據跑完轉彎後新的邊界相對位置決定如何貼著邊界走閃電型的路徑，以讓未被確定的線段有更大的行走空間。



III. “口對口”或 長方形區域繞線

長方形區域出現於 bump 之間，如右下圖所示的紅色區域。由於是封閉的區域，上下兩邊的需要連接的點，數量相等而且順序必定相同，否則必定會交錯而無法有正確的解答。此部份所要考慮的便是如何在有限的空間內讓所有的起點都能連到正確的終點。

考慮接鄰 bump 的部份為「短邊」，不失一般性將短邊擺在左右，考慮上下連接的情形（左右連接可以用同樣的邏輯完成，只須改變程式中 x ， y 值及比較大小的部份即可）。以下方的點為「起點」，上方為「終點」。則可以由右邊的點開始，一條一條連接。由底下的演算方式可知，每一條連接時只須考慮前一條連接的情形即可。



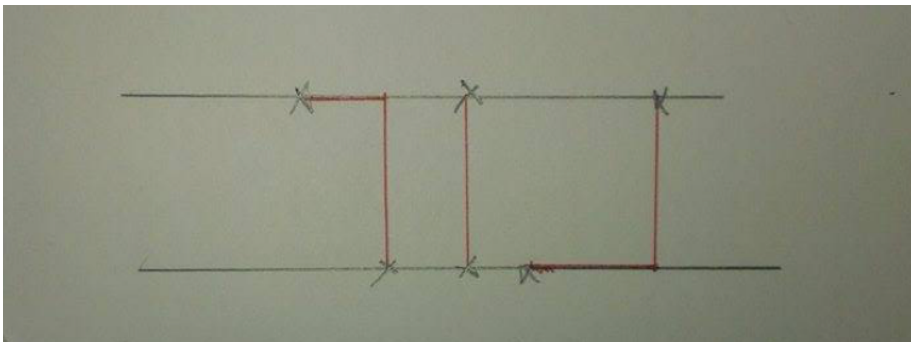
第一條線：

第一條線的終點如果在右邊，就先往右到底，再直接連上去

如果在上面，就向上連

如果在左邊，就先向上到底，再往左，

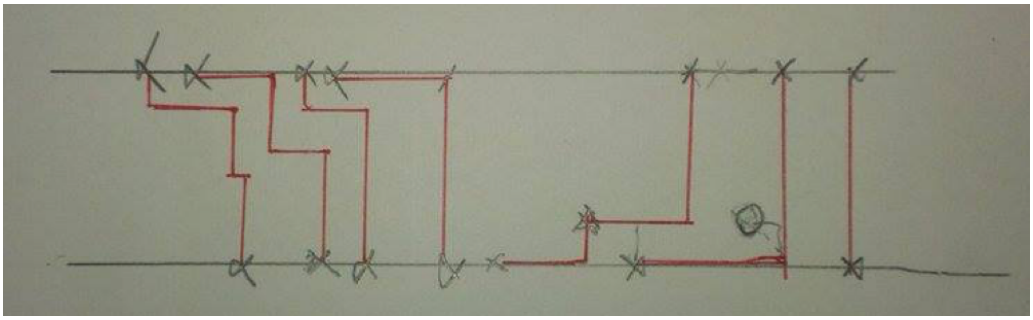
下圖顯示三種情形：



第二條線以後：

如果向右，判斷前一條線段的各個點往左一定間隔後，是否在終點的左邊，如果在終點的左邊，便需要沿著前一條路的路徑走，直到不在受影響。如下圖，第一條線完全不影響第二條線，而第二條線則在第一個點有影響，因此需要沿著走到第一個點的左上角，第二個點不影響。

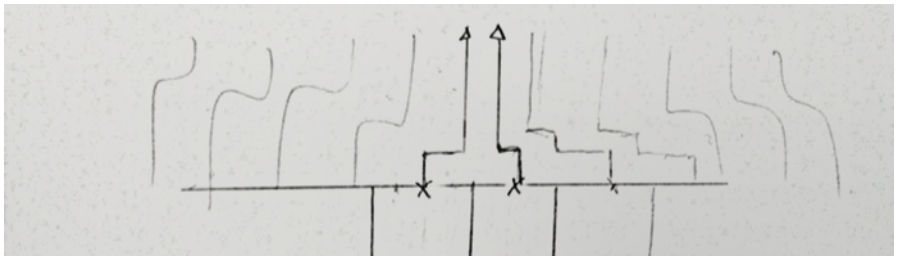
如果向左，便須在畫線段各個點時，判斷前一條線段的各個點往左一定間隔後，是否在當下那個點的左邊。如左邊數來第三條線。首先判斷出右邊線段第三個點之後，向上到第三個點的下方一定間隔。之後在沿著前一條的邊緣畫線。



IV. 從 driver 連到內部矩陣區域

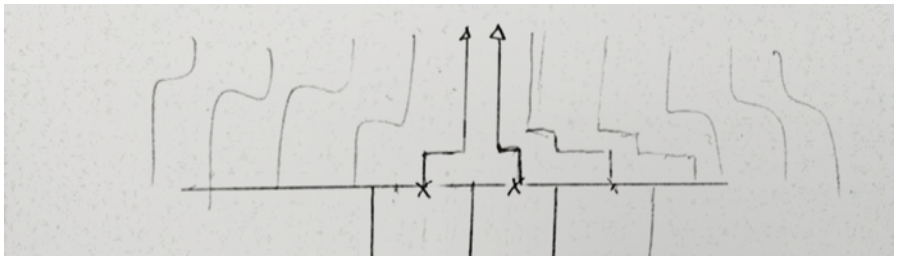
設計模型：我們假設 driver 的分佈如同 test case 144 的情形，driver 按照順序緊密排列在 bump 的外圍，而且在分區之後 driver 照順序排成一排。不失一般性，我們假定 driver 在底下，bump 分佈在上方。由 driver 連到 bump 最下層的第一條線必定會照順序排列，因為只有繞過別的 bump 才能改變順序。因此這部份與上一步部份相當相似，都是起點彼此在同一個水平線上，終點在另一個水平線上，如何從起點連到終點的問題。此外，由於外圍的 driver 左右橫跨的距離較大，因此右邊的 driver 到 bump 大多是往右下往左上的連線，而左邊的 driver 則是左下往右上的連線。

為了達成最緊密的線路設計，不影響其他區域的繞線情形。我們考慮讓所有往同一方向畫的線一次一起畫。相鄰兩條線段，如果右邊的要往左而左邊的要往右，那這兩條線段不會互相影響，因為他們的終點必須保持適當的間距。(如右圖)



反之，如果左邊的要往左而右邊的要往右，則明顯不會彼此干擾。因此我們設計讓所有附近都往左畫的線段一起畫，所有往右的線段一起畫，便能很簡單的設計出最節省空間的繞線方式。

在實際繞線時。往左畫的區域會找到最後一個（最左邊）往左畫的線條，先盡可能往左畫，接下來依次往右，貼著最邊界的地方走，直到走到終點底下。往右畫的區域也用同樣的道理，從最右邊要往右畫的線先畫，再依序畫完剩下的線段。



此演算法限制：

1. 我們的演算法會依據相對位置依序將 bump 與 driver 做 global routing，因此如果有不照順序的出現，我們的演算法永遠是後面繞過前面，也不可能有 bump 向上一層繞路的情形，因此不能保證路徑最短，並且在一些極端狀況例如 test_144 的 bump 134,135，可能導致某些區域的 spacing 不夠承擔所有線寬而繞線失敗的情形。
2. cross section bump 的繞線不保證最短距離，因為我們在放置 virtual bump 的時候，並沒有檢查放置位置是否為讓距離最近的地方。
3. 我們在處理 detail routing 的時候也並未優化線到 bump 如何能最短，我們是固定位置給定。

與論文的比較：

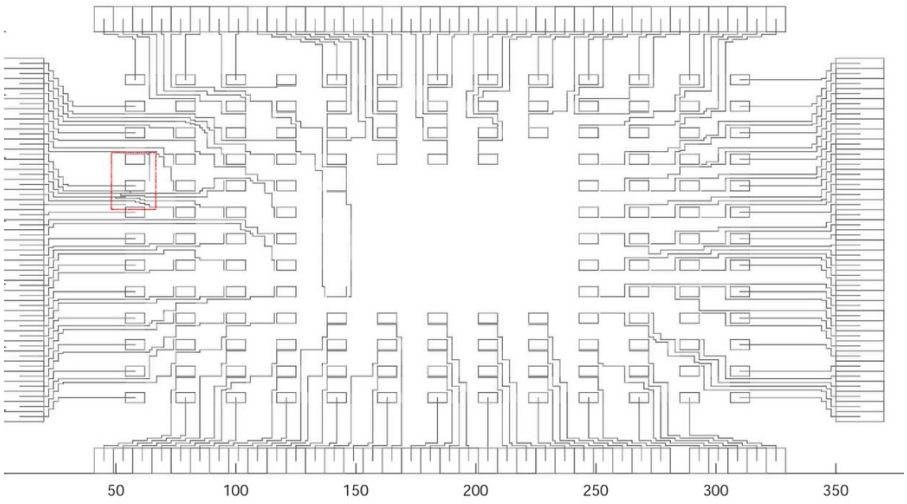
1. 繞線的部分在看完論文的處理方式之後，由於我們證明他們的方法一定繞得出來，所以我們重新想了一個。新方法具有 bump 順序不能太過混亂的限

制，因為這樣即便我們可以知道 global routing 的路線，但是很有可能導致空間不夠，而不符合題目的要求。不過我們的方法比較好理解。

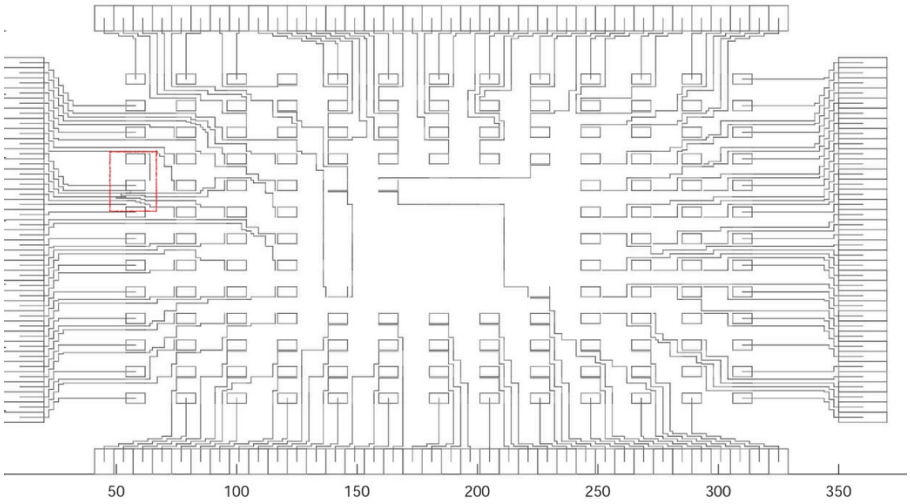
2. 論文處理 cross section 的方法跟我們一樣，將 cross section 的 bump 另外設立 virtual bump，放在該區域的最靠近圖中心的一層，將他們視為 bump 與 driver 的組合，用既有的方法繞線即可。與論文不同的是，論文終將 virtual bump 的 index 設為垂直對應的 driver 的 index-0.5，在 test144 及 test143 的 case 中會剛好對，因為該區的 driver 剛好照著順序排。而我們處理的方式並不需要 driver 照順序排好。
3. detailed routing 的部分，因為我們同樣無法證明論文中的方式在大多數情況下可以成功，因此 detailed routing 的部分是我們自己設計的。

結果

test143_b_input



test144_b_input



組員

	馬靖孟	葉津源	陳溥生	陳立馨
信箱	b02202019	b02202028	b02202044	b02202048
電話	0913821052	0972311499	0927376243	0921325581

參考資料

1. http://cad_contest.ee.ncu.edu.tw/problem_d/default.html
2. http://nthur.lib.nthu.edu.tw/dspace/handle/987654321/33079