逢　甲　大　學

資訊工程學系專題報告

VLSI Floorplanning With Abutment Constraint using Genetic Floorplan Algorithm

學　生：龔泰維　（四乙）

指導教授　：　陳德生 老師

中華民國九十二年十一月十三日

目錄

圖表目錄・・・・・・・・・・・・・・・・・・・・・・・・・II

摘要・・・・・・・・・・・・・・・・・・・・・・・・・・・・V

第一章 導論・・・・・・・・・・・・・・・・・・・・・・・・1

第二章 問題定義・・・・・・・・・・・・・・・・・・・・・19

第三章 GFA・・・・・・・・・・・・・・・・・・・・・・・・21

第四章 做法・・・・・・・・・・・・・・・・・・・・・・・28

第五章 測試結果・・・・・・・・・・・・・・・・・・・・・33

第六章 結論・・・・・・・・・・・・・・・・・・・・・・・36

參考資料・・・・・・・・・・・・・・・・・・・・・・・・・37

圖片目錄

Fig.1 VLSI Design Cycle • • • • • • • • • • • • • • • • • • • 1

Fig.2 Functional Flow • • • • • • • • • • • • • • • • • • • 2

Fig.3 Circuit Design • • • • • • • • • • • • • • • • • • • 3

Fig.4 Layout • 3

Fig.5 VLSI Design Flow • • • • • • • • • • • • • • • • • • 4

Fig.6 Physical Design Cycle • • • • • • • • • • • • • • • 5

Fig.7 Circuit Partitioning • • • • • • • • • • • • • • • • 6

Fig.8 Floorplan Example • • • • • • • • • • • • • • • • • 6

Fig.9 Placement Example • • • • • • • • • • • • • • • • • 7

Fig.10 Routing Example • • • • • • • • • • • • • • • • • • 8

Fig.11 Slicing Floorplan • • • • • • • • • • • • • • • • • 10

Fig.12 Slicing Tree • 10

Fig.13 Non-slicing Floorplan・・・・・・・・・・・・・・・・11

Fig.14 表示法比較 ・・・・・・・・・・・・・・・・・・・・・・13

Fig.15 GA Flow・・・・・・・・・・・・・・・・・・・・・・・15

Fig.16 Abutment Constraints Define ・・・・・・・・・・・19

Fig.17 PE example ・・・・・・・・・・・・・・・・・・・・・21

Fig.18 GFA Flow ・・・・・・・・・・・・・・・・・・・・・・23

Fig.19 Crossover Flow ・・・・・・・・・・・・・・・・・・・25

Fig.20 Merge Rule ・・・・・・・・・・・・・・・・・・・・・28

Fig.21 Swap to merge・・・・・・・・・・・・・・・・・・・・29

Fig.22 Merge Flow ・・・・・・・・・・・・・・・・・・・・・30

Fig.23 GFA crossover with Abutment constraint ・・・・・・31

Fig.24 Ami33 的結果・・・・・・・・・・・・・・・・・・・・34

Fig.25 Ami 49 的結果 ・・・・・・・・・・・・・・・・・・・・・・34

Table.1 測試結果・・・・・・・・・・・・・・・・・・・・・・・33

Table.2 與 CBL 結果之比較・・・・・・・・・・・・・・・・・・35

摘要

在 VLSI Physical Design 中許多常見的 constraints 有助於 block 之間資料的傳遞，而 abutment constraint 是其中之一。這種控制數個有限制條件的 module 的相關位置的問題，很不好解決。利用 GFA 可以在較短的時間找到一個好的 floorplan 的優點，來試著解決 abutment 的問題。最後藉由數種不同 benchmark 的實驗結果，可以證明這個方法不但有效，而且可以快速的找到一個好的解。

第一章　導　論

VLSI 設計按照著一定的流程。從 System Specification、Architectural Design、Functional Design、Logic Design、Circuit Design、Physical Design、Fabrication 到 Packaging 共八個步驟。如 Fig.1 是 VLSI Design Cycle。

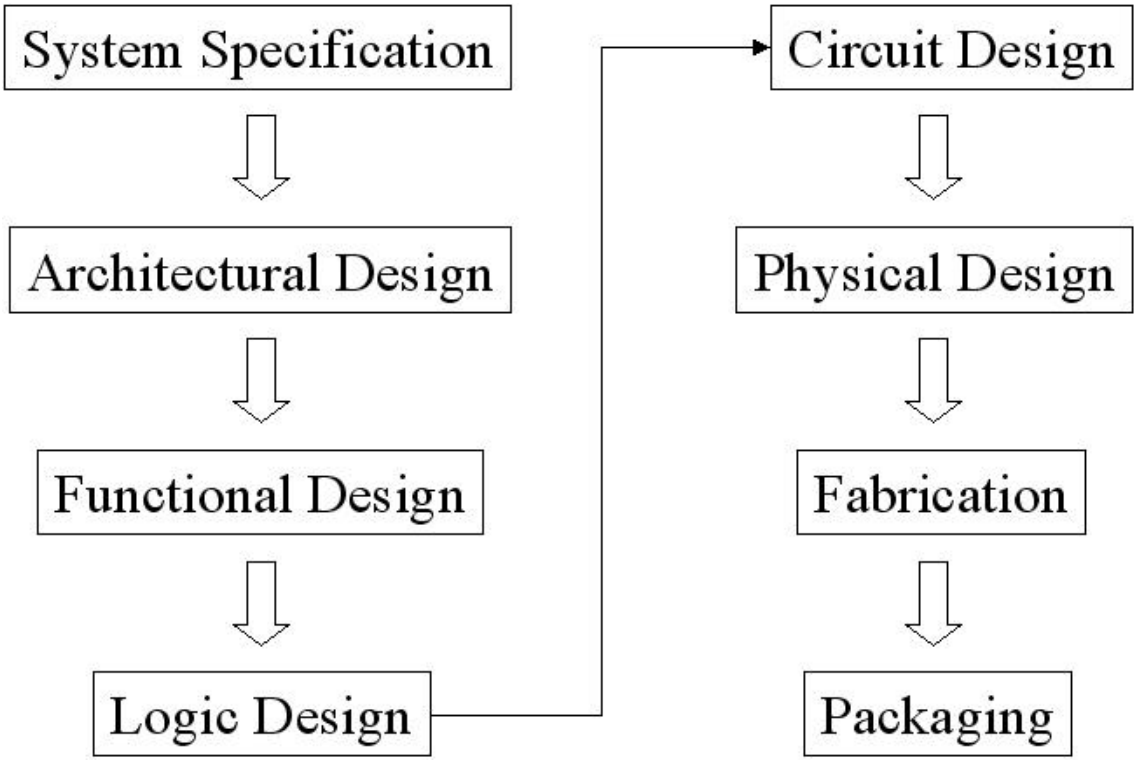

Fig.1 VLSI Design cycle

其中, System Specification 詳細說明整各 VLSI System 的 size、speed、power 以及功能。

Architectural Design 決定系統的架構，例如：RISC/CISC、ALU 的數量、pipeline 架構、Cache 大小…等等。這些決定都會對整各系統的效能、die area、耗電量…等等提供一些精準的判斷。

Functional Design 對各個功能性單元做定義，並且以流程圖定義他們之間的關聯，如圖 Fig2。

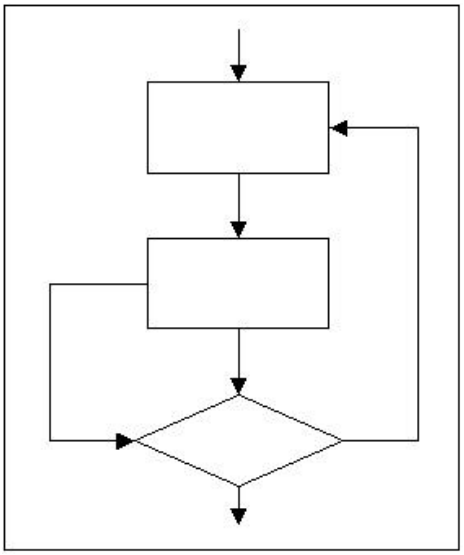

Fig.2 Functional Flow

Logic Design 對布林代數表示法、控制流程、暫存器配置…等等的邏輯做設計，稱為 RTL(Register Transfer Level。RTL 用 HDL (Hardware Description Language)來表示，如 VHDL 、Verilog。

Circuit Design 設計整各電路圖，如 Fig.3。包括：電晶體跟內部連結…等等，結果稱為 netlist。

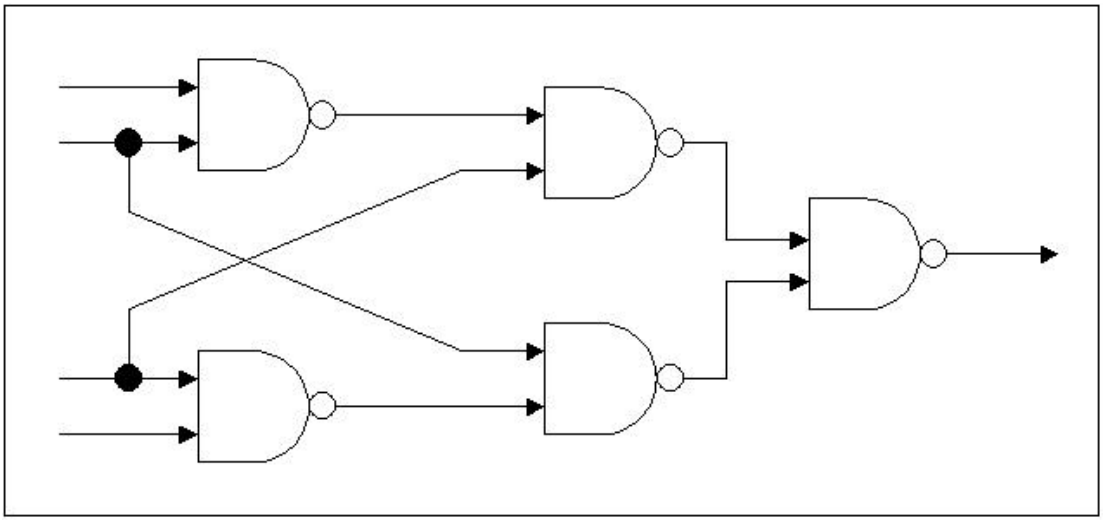

Fig.3 Circuit Design

Physical Design 把 netlist 轉換成幾何的表示法，結果稱為 layout。如 Fig.4。

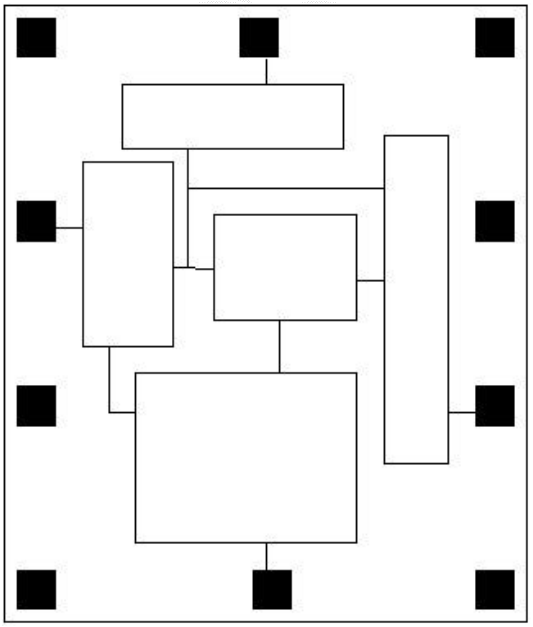

Fig.4 Layout

Fabrication 把結果經過 lithography, polishing, deposition, diffusion…等等的處理後，產生一個晶片(chip)。

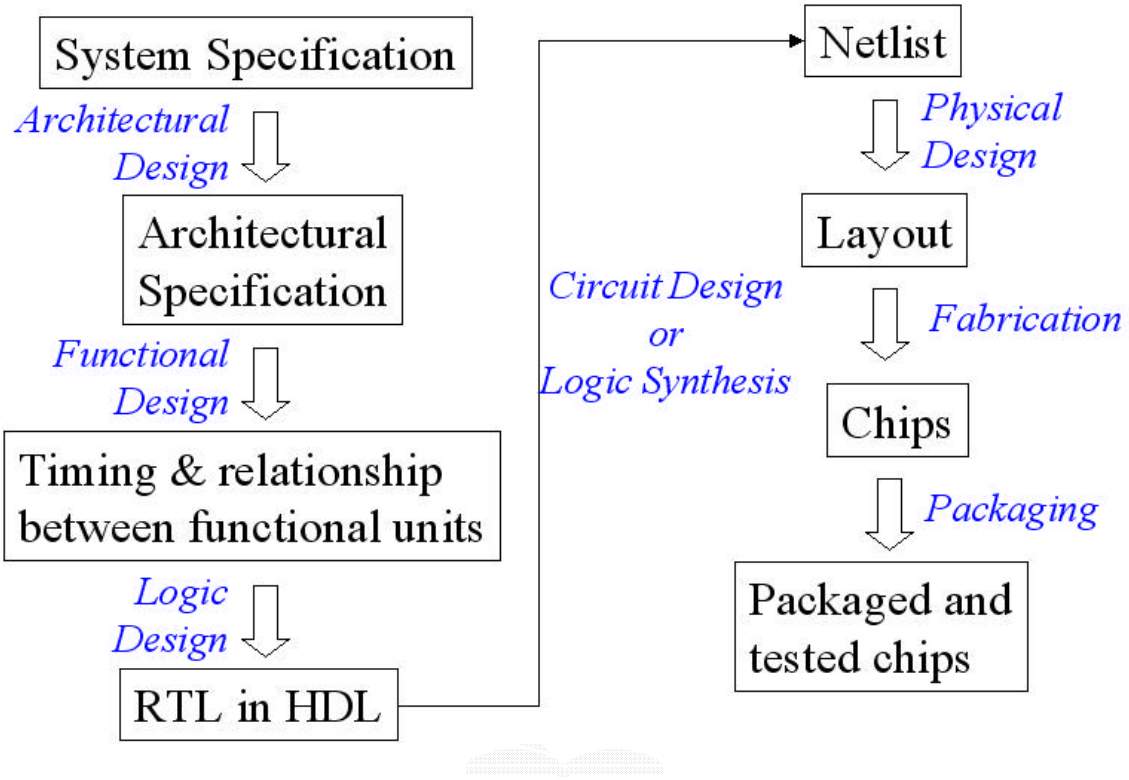

Fig.5 VLSI Design Flow

Packaging 把晶片放在 PCB (Printed Circuit Board) 或是 MCM (Multi-Chip Module)。便完成了整各 VLSI Design。而整個 Design 的各個步驟以及結果的名稱如 Fig5。

在 VLSI 設計中 Physical Design 佔很重要的地位。而 Physical Design 包括了 partition、floorplan、placement、routing…等等。

如 Fig.6 就是整個 Physical Design Cycle。

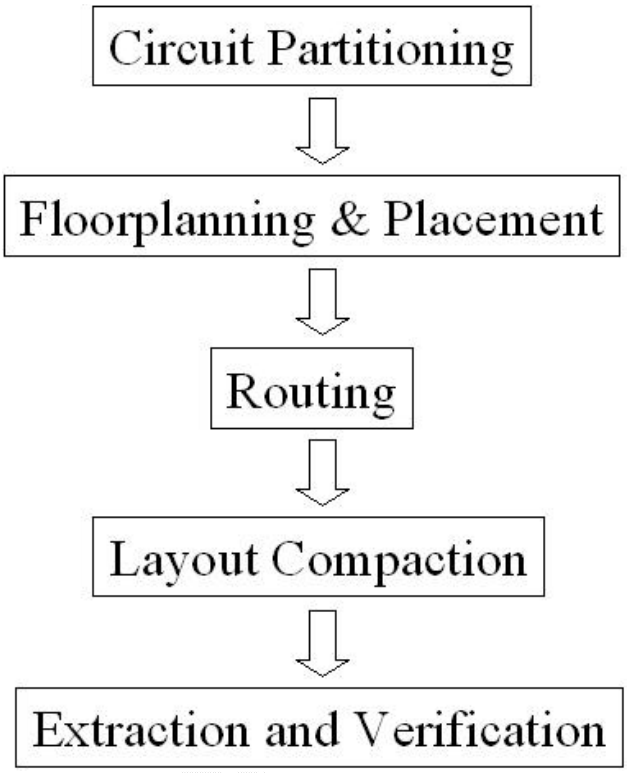

Fig.6 Physical Design Cycle

Physical Design 第一步就是 Circuit Partitioning 。這時把整個大電路分割成數個不同的子電路稱為 block，如 Fig.7 分割成三個 block。而在分割時要考慮到例如：block 的個數、block 的大小還有 block 之間的連線…等等。這些都會影響到之後的其他步驟，和整個設計的效能。

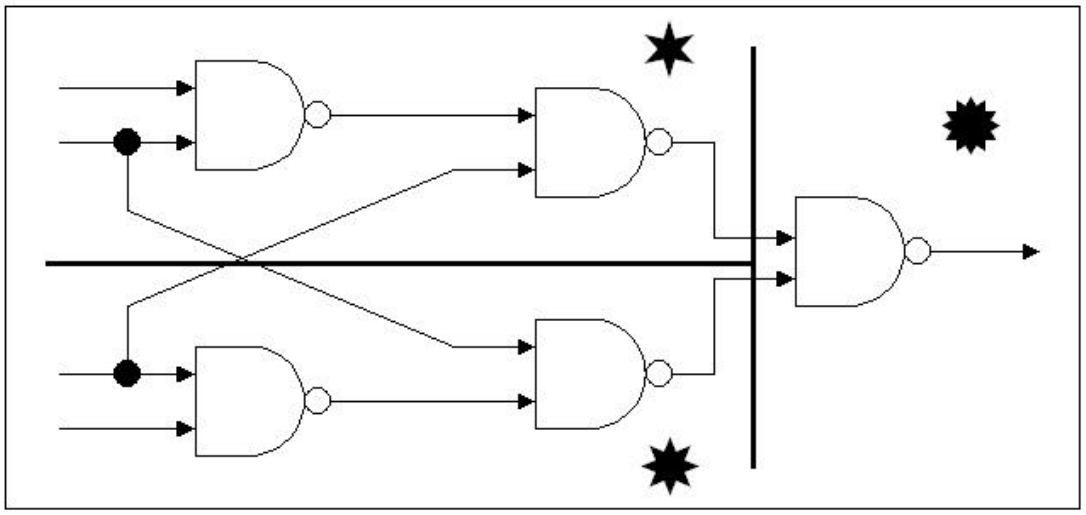

Fig.7 Circuit Partition

Floorplanning 設計產生一個好的 layout，放置所有 block、功能性單元…等稱為 modules 的位置。這時 module 的形狀、面積以及 I/O pin 腳的位置…等等還不是固定的。

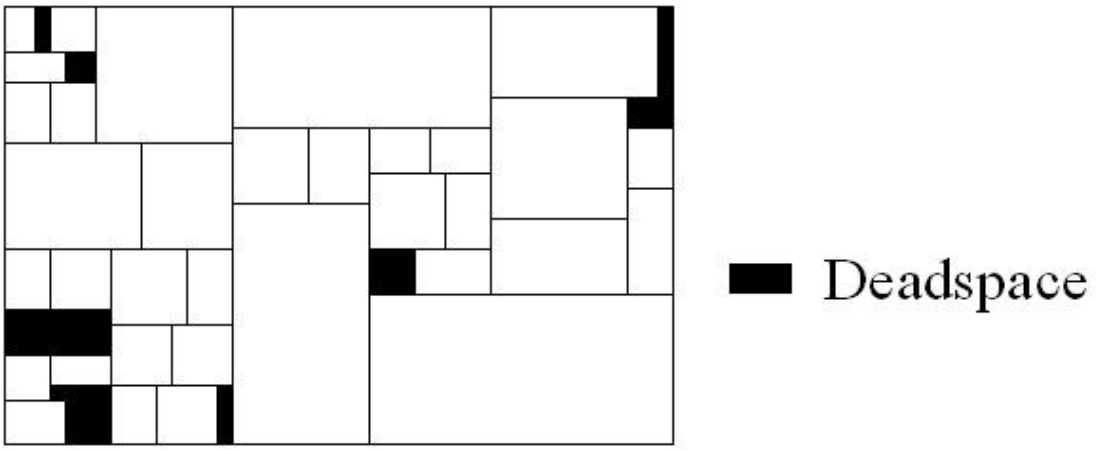

Fig.8 Floorplan Example

Placement 跟 floorplanning 很像，而這時的 module 是 gates 或 standard cells…等形狀大小面積都已經固定的。確切的佈置每一個 modules，跟 floorplanning 一樣，最主要的目標都是 delay、總面積以及 interconnect cost 作最小化。

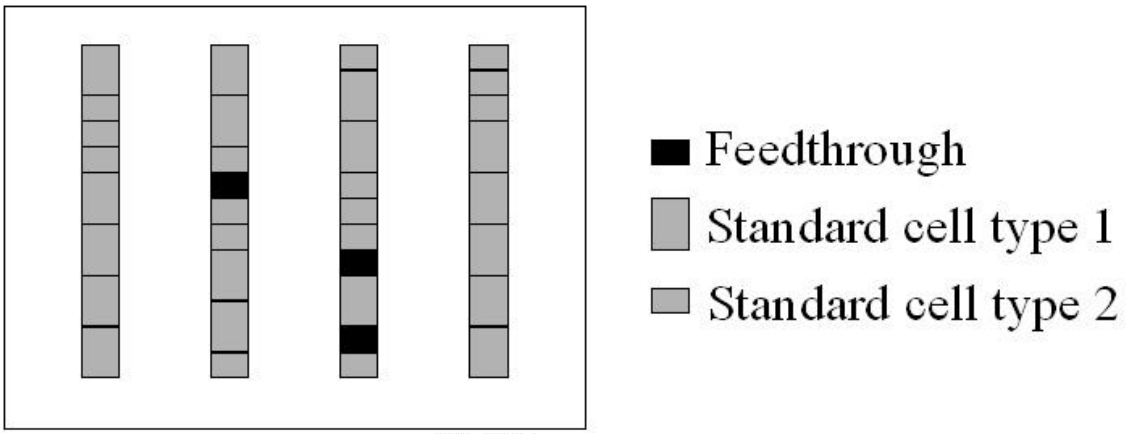

Fig.9 Placement Example

Routing 是要完成所有 modules 之間的線路連結，routing 包含 global routing 和 detailed routing 。在 routing 的時候要考慮到 critical path、clock skew、wire spacing…等因素求最佳化。

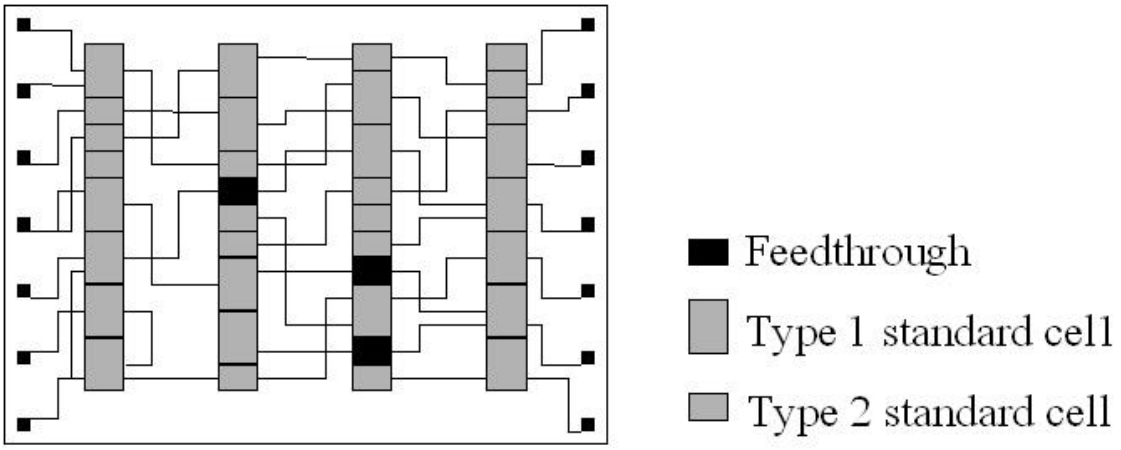

Fig.10 Routing Example

Compaction 把 layout 的結果作各個方向的壓縮，以求整個 chip 面積的最小化。

Verification 在檢查 Layout 的結果是否正確。先做 DRC(Design Rule Checking)，確定沒有違反 Layout 的規則。接著 circuit extraction 由 layout 產生一個電路，來跟一開始的 netlist 比較正確性。最後 performance verification 由 extract 出來的資訊，計算整個的 delay 跟電容…等等。

整個 Physical design 中，由於 Floorplan 結果的好與壞會影響一個晶片的面積成本與接線成本，因此 Floorplan 扮演非常重要的角色。一個好的 Floorplanner 除了要能找到最佳化的解之外，還要能夠解決各種特定的 Constraints。例如有時或許會因為某些特定的目的，而希望把特定的 block 給放在另一個的 block 的旁邊，或者是希

望 Floorplan 的結果不要超過某個限定的範圍大小…等等。設計一個好的 Floorplan 可以解決每種不一樣的 Constraints 的演算法是一件困難的事。

現已知有數種演算法可以來解 Floorplan 的問題，其中最有名、被最多研究者使用的是 Simulated Annealing（SA）演算法。在之前 SA 被認為是解 VLSI-CAD 問題，最容易發展、而且能夠得到好解的方法。雖然 SA 能得到不錯的解，但是這個演算法執行起來十分費時。比 SA 發展更早的 Genetic Algorithm（GA，基因演算法），直到最近才開始廣泛的被運用在各種 NP 的問題上，其中也包含 VLSI-CAD 的問題。GA 所採用的是仿自然界競爭演化的方式來達成最佳化的目的。但是原始的 GA 雖然是以基因及演化的方式來動作，但是並沒有將親代的優良基因有效的遺傳到子代上。

而 Floorplan 分成兩大類：分別是 slicing 和 non-slicing。Slicing 的 Floorplan 可以被遞迴的依垂直和水平的方向切割，而一個 Slicing Floorplan 可以表示成 Binary Tree，稱之為 Slicing Tree，如 Fig.11。

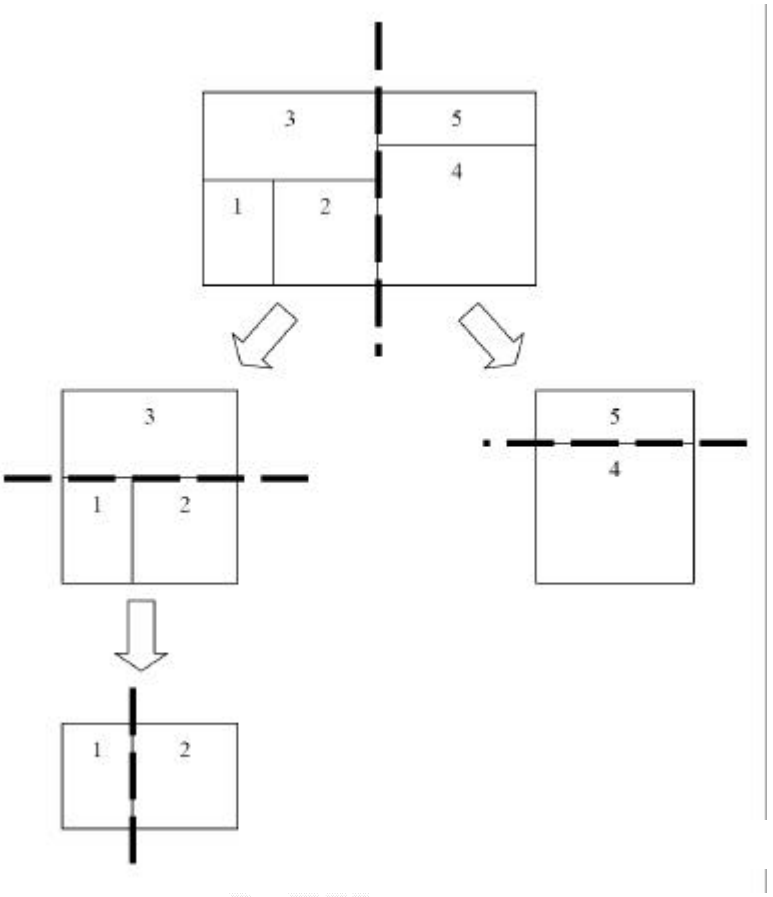

Fig.11 Slicing Floorplan

Fig.12 是它的 slicing tree。

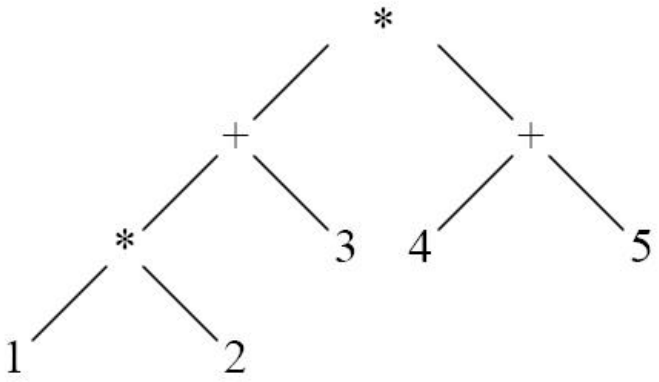

Fig.12 Slicing Tree

Non-slicing Floorplan 顧名思義就是無法分割的 Floorplan，如 Fig.13。

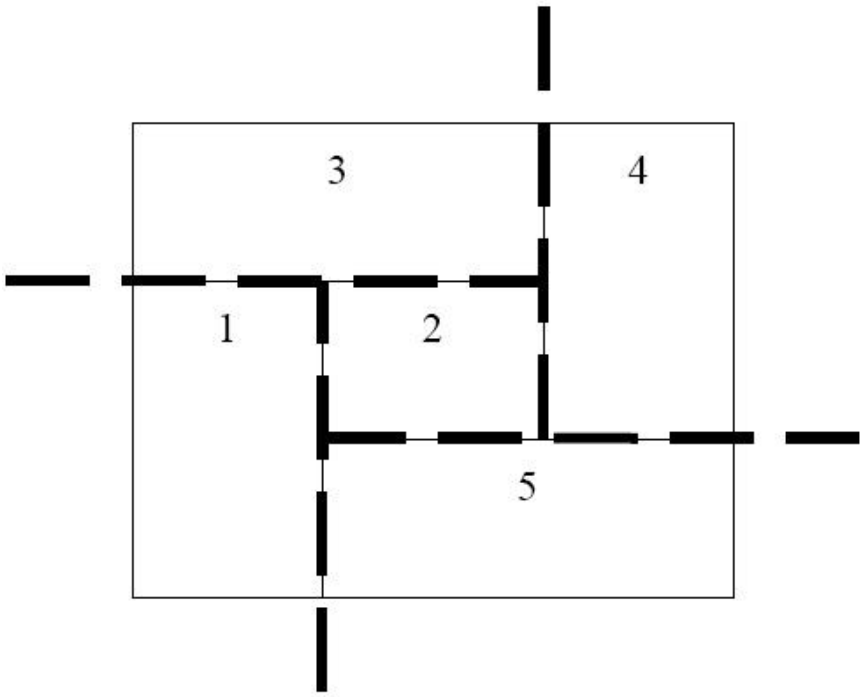

Fig.13 Non-slicing Floorplan

用各種演算法來解 Floorplan 的問題。首先就是把演算法的結果，用表示法來表示。現在有許多種的表示法如 Binary tree 、Sequence Pair(SP)[7]、Bounded slicing grid(BSG)[9]、O-tree[8]、B*-tree[4]…等等。Binary tree 只能表示 slicing floorplan，Tree 的終端節點為 module，中間節點為水平或垂直的結合方式。SP 用兩條序列表示 module 在 X、Y 軸上的相對位置，由兩條序列來知道各個

modules 在 floorplan 中的位置。BSG 將一個平面，以相同長度的水平和垂直線，分割成數個相同大小的空間，每個空間最多只能放進一個 block，藉此得知每個 block 彼此相對的關係位置。O-tree 非二元樹，可以一各節點有多個子節點，而它的 floorplan 中沒有任何一個 block 可以移動，也就是 X Y 座標都是已經固定的，而子節點的位置在於父節點的右邊，然後再以另一個值來表示其垂直的位置。B*-tree 以 binary tree 為基礎，它的 root 是最左下角的 module，然後節點的左右子節點可以表示它的上面或右邊的 modul，依照方向的不同可以分為水平或垂直的 B*-tree。Fig.14 列出這些表示法的優缺點[4]。其中除了 Binart tree 只能用在 slicing floorplan 之外，其他的都可以同時解 slicing 和 non-slicing。而現在也有許多其他的表示法被提出，目的就是要改善這些原本的表示法。

Represent.	Advantages (a) and Disadvantages (d)
Binary tree	a1. efficient a2. flexible to deal with hard, pre-placed, soft, and rectilinear modules, etc a3. smaller encoding cost a4. can operate on the tree directly, no need to do transformation during processing a5. can evaluate area cost incrementally a6. transformation between representation and placement takes linear time
	d1. can handle only the slicing structure
Sequence pair/ BSG	a1. can handle non-slicing structure a2. very flexible in representation
	d1. time-consuming d2. the solution space is large d3. sequence encoding cost is high d4. harder to transform between a sequence pair and a placement d5. sequence pair cannot handle soft modules directly d6. BSG incurs redundancies
O-tree	a1. can handle non-slicing structure a2. the solution space is smaller a3. transformation between representation and placement takes only linear time a4. encoded by fewer bits than sequence pair and BSG
	d1. less flexible than BSG/sequence pair in representation d2. tree structure is irregular, harder for implementation d3. need to encode and operate on module sequence d4. need to transform between the tree and its placement during processing d5. inserting positions are limited, might deviate from the optimal during solution perturbation
B*-tree	a1. can handle non-slicing structure a2. binary-tree based, efficient a3. flexible to deal with hard, pre-placed, soft, and rectilinear modules, etc a4. smaller encoding cost a5. except for handling soft modules, only need to transform from a tree to its placement during processing, which takes only linear time a6. can evaluate area cost incrementally a7. the solution space is smaller
	d1. less flexible than BSG/sequence pair in representation

Fig.14 **表示法比較**

其中一個表示法就是polish expression(PE), 這是改良於Binary tree 的表示法。把 Binary Tree 作後序走訪就是 PE，把表示法由樹的結構改為陣列，後面會詳細定義。PE 也是解 Slicing 的 Floorplan。使用 Slicing Floorplan 有幾個好處：

(一)因為只在 Slicing Floorplan 中找解, 所以相對的 solution space 就會因此而變小，所以會使得找解的時間比較短。

(二) Slicing Floorplan 可以充分的利用 Soft modules 的可調性，得到 block 結合比較緊的結果。

利用這些特性，也使得在處理包含各種 Constraints 的問題時，更容易得到一個不錯的解。

GA 一開始會先隨機的創造一群初始值，稱之為族群（Population)。在族群中每一個個體稱為染色體（Chromosome)，染色體是一串符號，通常由二進制的字串所表示，而每一個符號都叫做基因。在求解的過程中，每個染色體代表著一個解。另外有一個評估函式（fitness)，來判斷每個解的好壞。接著從族群中找出兩個個體作為親代，根據類生物的基因運算之後，產生一個子代，稱為offspring。再把產生出來的子代，根據突變機率(mutation rate)來決定是否產生突變 。然後再經由評估函式判斷子代的好壞，來決定是否要取代掉較差的親代。按照這樣的流程運作一次，稱為一個世代。

每個世代之後，族群裡染色體的數目都要保持一定。當經過了幾個世代之後，族群裡不好的親代，漸漸的被所產生出來比較好的子代給取代。最後，族群裡最好的一個個體，就是求最佳化的一組解。

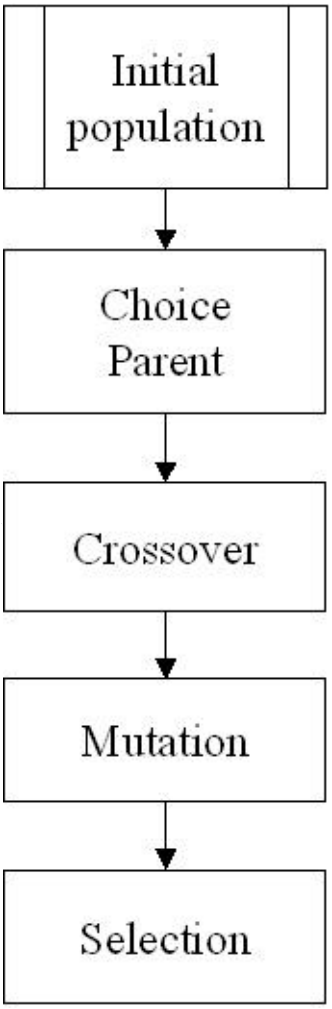

Fig.15 GA Flow

在基因演算產生子代的過程中，最重要的一個步驟就是交配（Crossover)。Crossover的行為模式就是把兩個親代中擷取部分的基因，在把這些基因結合成一個新的子代。最簡單的方式就是隨機的將

親代切完左右兩半，由一個親代各提供一半。子代就是由一個親代提供左半部，另一個親代提供右半部所組成的。這樣的Crossover適用於二進制的表示法，當表示法不同的時候，使用的Crossover的機制也相對的不同。Crossover最基本的目的就是要讓子代可以繼承兩個親代各有的優良基因，讓子代可以比親代還要優良。

每個親代被產生出來之後，還會根據突變機率來決定是否要作Mutation。因為每個子代都會繼承親代的優點，而且比較優良的親代被挑選出來的會也比較高，所以在過了幾個世代之後，會使得族群裡的所有染色體趨近於相近。因為優秀的特徵一值被遺傳下來，但是比較差的部分也是沒有被取代掉，所以這時候solution space就會變的比較狹小。所以，為了改善這種情形出現的太快，便加入Mutation的機制來擴展solution space，以便能夠藉此搜尋到更多其他的可能解。

Mutation並不會直接產生一個新的子代。它只會對crossover產生後的子代，把子代作隨機的改變。最簡單的做法就是，從子代的基因中，隨機的挑選兩組作交換。這個步驟在基因的演算法中，扮演著幕後重要的角色。因為它雖然不會直接產生一個子代，但是它隨意的讓基因組合產生改變，讓染色體出現不同於族群裡其他染色體的排列組合。這樣的結果，使得族群裡的所有染色體，不會太快就全部趨近於相似，藉此將solution space擴大，而是否作Mutation也是藉著一定

的機率來判斷是否會產生。當Mutation做完之後，才會進行選擇的機制，來判斷是否要取代掉原先的兩個親代。

當產生新的子代之後，就會需要一個選擇的步驟。以便從所有的親代中和新產生的子代中，來決定下一個世代的族群，而且要維持族群一定的數量，再從新的世代中再選擇出產生下一世代的親代。目前所使用的選擇方式有很大的差異。下面列出三種：

(1) Competitive selection：在親代及子代中挑出P個做取代（P表示族群大小）

(2) Random selection：按照機率做隨機選擇要取代的親代。這個方法有時是有好處的，因為這個方法可以使族群中個體的差異性保持較久，避免太快趨於相似。若純粹使用competitive selection，整個族群會快速的收斂於一個解，這個解和族群中的其他解相差很細微。如此一來就會喪失找到更好的解的能力。但是，相對的這個方法也會不小心將比較優良的親帶也給取代掉，所以又有第三個方法。

(3) Stochastic selection：將最好的保留下來，剩下的以random的方式決定是否留下。這個方法包含了competitive 和randomness。一方面能確保整體優良性的的上升，而且永遠不會漏掉最好的解，因為好的解優先保留，不會參與random

選取。而random 選取的機率和每一個個體的fitness 成比例。換句話說，就是fitness 的值越好的被選到的機率越大。

當選擇取代完之後，新的族群就產生了。然後再反覆的執行多個世代，等到整各族群漸漸的趨於相似，便取其中 Fitness 值最高的就是求最佳化的解。

針對 GA 沒有遺傳親代的優良基因的問題，所以提出了改良的 GFA。這份報告就是以 GFA 為基礎並且加上處理 Abutment constraints 的功能。abutment 是指在求 floorplan 的過程中，要求某些 block 因為某些特殊的關係，而必須要依水平或是垂直的方向，一個 block 和一個 block 緊緊的相鄰連接在一起。而藉著 GFA 可以把某些必須做垂直或水平結合的 block，不允許它以不符合限制方向進行結合。因為是 slicing 的 floorplan，所以在這時便可以輕易的避開即使按照不符限制的方向結合，也有可能最後結果會正確的情況。雖然這樣一來，所得到解的可能性就變小。但是相對的也縮小的找解的時間性，以及驗證所得到的解是否符合 constraints 限制的時間。接下來的幾章，會對 Abutmant Constraints 作詳細的說明和定義，還有詳細的介紹核心引擎的 GFA。第四章會說明實際的做法。第五章將會看到最後實驗的結果，以及和別人做出來的結果相比較。最後，也對此作一個結論。

第二章　問題定義

在 Floorplaning 中每個 block 由（h_i , w_i）所定義，而h_i、 w_i 分別表示一個 block 的高和寬 每一個 block 定義它的 aspect ratio 為 h_i/w_i 。Block 又分為兩種：Soft block 和 Hard block。Soft block 有固定的面積 但是 aspect ratio 是可以改變的, 而會定義 Soft block 的 aspect ratio 在一個範圍之內。Hard block 有固定的面積和固定的 aspect ratio 它是沒有辦法改變它的寬高。因為 soft block 的可變性，所以 soft block 也可以增加解的 solution space。

而 Abutment constraints 的定義是，如 Fig.16 假如 block A 高 ha 和 block B 高 h_b。並且 block A 和 block B 作水平結合, 而兩個 block 之間相鄰的長度大於等於比較小的ha、 h_b，那麼 block A 和 block B 就是水平的 abut。反之，若是垂直的 abut 也是這樣的定義。

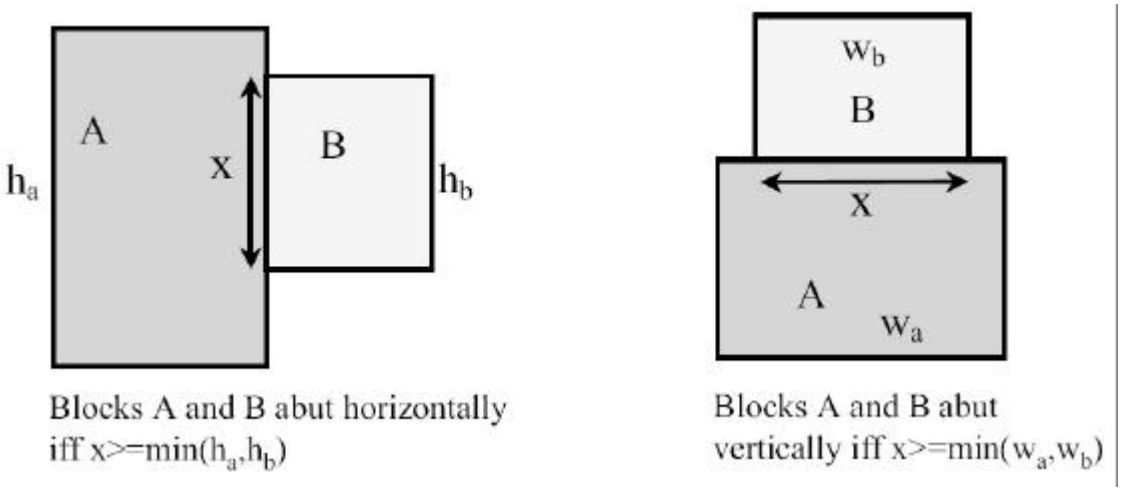

Fig 16 Abutment Constraints Define

定義所有的 block 皆為 soft block, 在符合所有定義的 abutment constraints 的同時，要找到一個總面積最小的解，並且要盡量的降低找到最佳解的時間，就是這報告所要達到的目標。

第三章 GFA

GFA（Genetic Floorplan Algorithm）以GA為基礎，採取GA的優點要來解Floorplan Optimal。在這選擇的表示法是Polish Express (PE)。

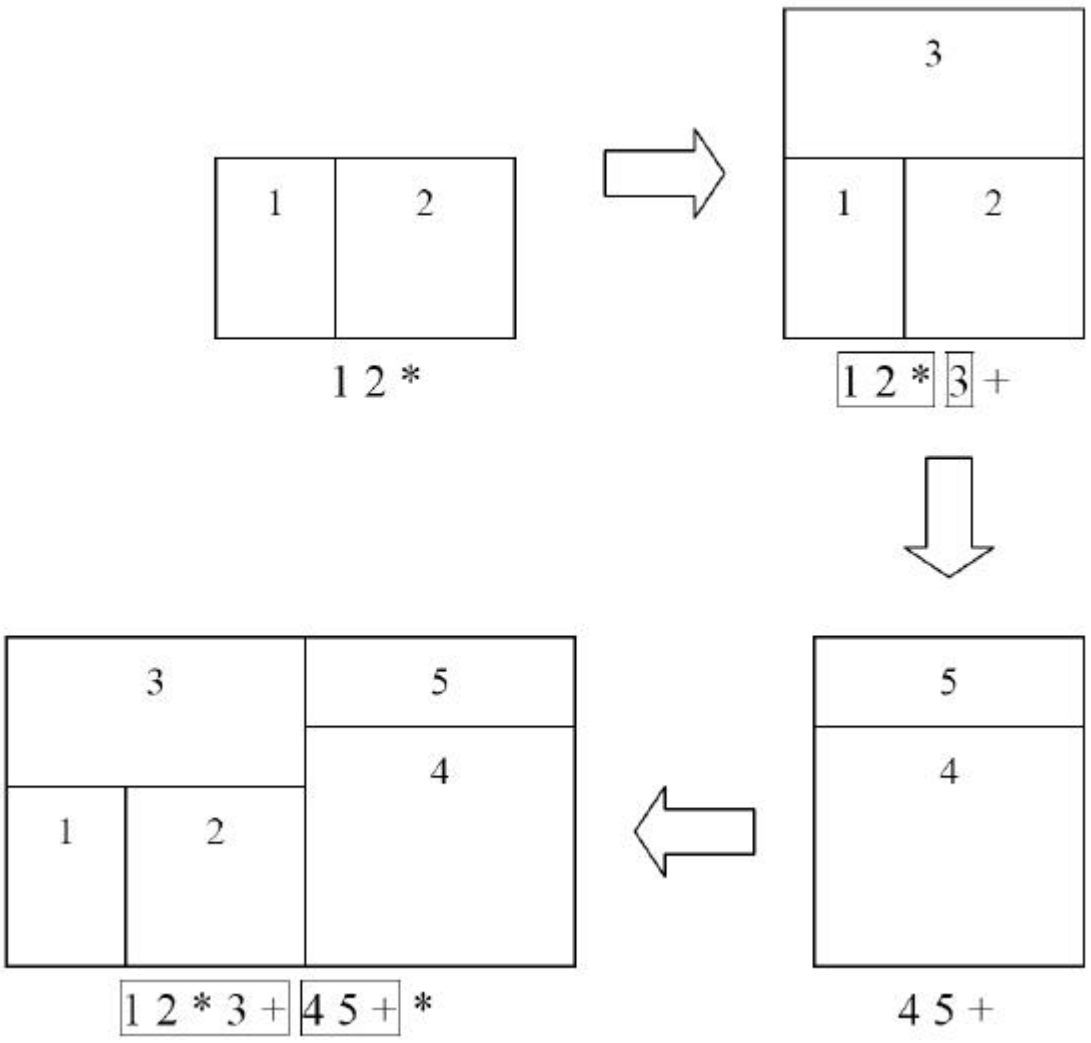

Fig 17 PE example

PE的表示法中，有三種符號：module的標號、「＊」和「＋」。「＊」表示水平結合，「＋」表示垂直結合，以Fig.17為例子來說明：１２＊３＋４５＋＊。

（1 2 *） 即表示Block 1 和 2 做水平結合。再來把 3 跟結合好的 1 和 2 垂直結合為 （(1 2 *） 3 +)。然後block 4 、 5 做垂直結合，為 （4 5 +)。最後再把兩者做垂直結合為 (((1 2 *) 3 +)(4 5 +) *)。由此可知用Polish Express表示的為Slicing floorplan。

GFA便以PE為chromosome表示法，而每一條Chromosome都表示一個slicing flooroplan的解。而所用的Fitness Function為：

$$\text{Fitness(p)} = \frac{Floorplan_{Area} - Block_{Area}}{Floorplan_{Area}}$$

其中$Block_{Area}$是指所有Block的總面積，而$Floorplan_{Area}$是指由polish expression所表示出整各Floorplan的面積。這個fitness的值表示dead area在整各Floorplan所佔的比例，當值為 0 表示完全沒有dead area。反之fitness越大的話，就表示dead area越大，也就是說整個Floorplan的面積越大。所以Fitness值越小的結果就是越好的。

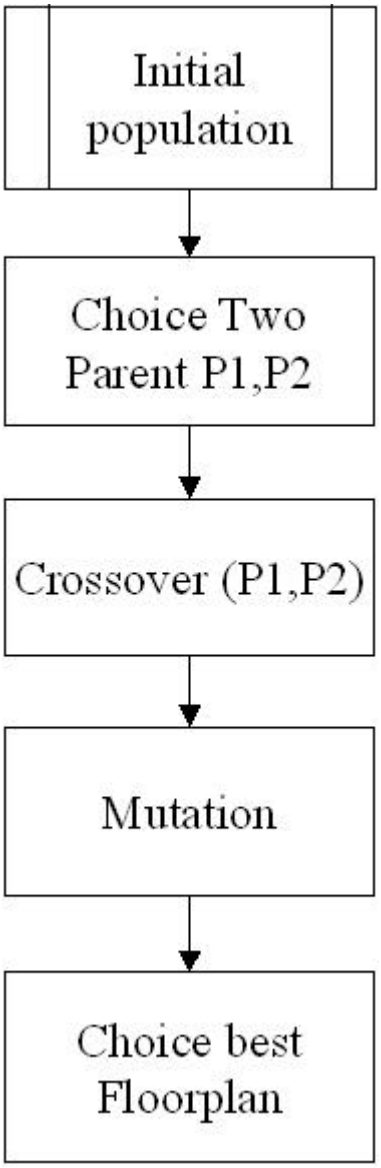

Fig.18 GFA Flow

如Fig.18。GFA一開始先亂數產生一個族群，接著在初始族群中，隨機的挑選出兩條不一樣的Chromosome 作為親代。兩條親代便開始做Crossover的步驟。完成後接著進行Mutation的步驟。而一開始Mutation的發生機率較小。等到族群中Chromosome開始趨於相近，比較沒有變化的時候，就會把Mutation rate適當的加大。使得Mutation

的作用明顯一點，藉此避免solution space過於快速的收斂。使得搜尋結果可以跳脫local optimum。當以上兩個步驟完成後，由fitness來判斷是否取代掉原本結果較差的親代，便完成一個世代。經過了幾個世代之後，要是所有的子代都無法比親代優良時，就表示子代無論如何都無法再取代親代，因此就達到了演算法的停止條件。這時族群中最好的Chromosome就是所找到的最佳解。

在整個過程中Crossover就是最重要的角色，因為它是產生負責產生子代的步驟，而每種表示法都需要不同的Crossover方式。對應Polish Express，Crossover總共有八個步驟 而在其中除了用fitness判斷親代中優良的基因之外，還有另外一個threshold值來控制Crossover的動作。在Crossover一開始，threshold初始值為 0 ，而在Crossover的過程中會慢慢的加大threshold值。一但fitness小於threshold值的時候，便表示這基因不被接受。由於fitness代表著dead area，所以也就是說在Crossover的一開始，並不容許任何的dead area。直到後來才會漸漸的容許dead area的存在，也藉著如此來得到一個最小面積的Floorplan。這八個步驟如Fig.19，說明如下：

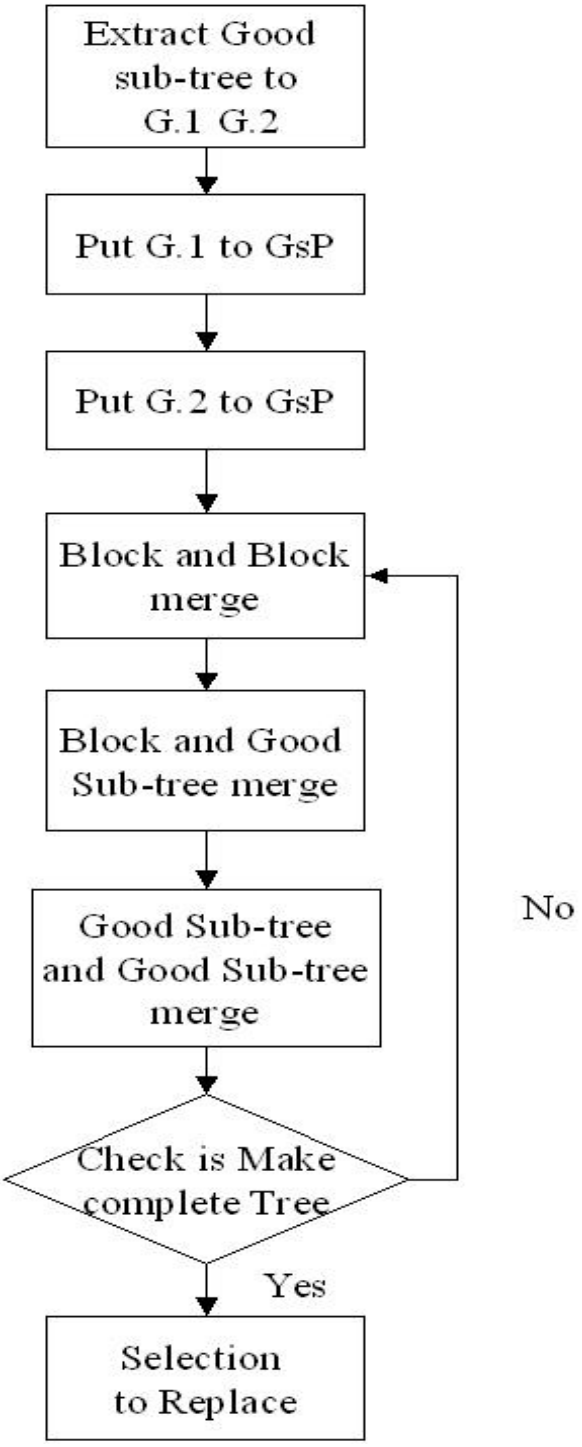

Fig.19 Crossover Flow

Step.1：從兩個親代中，找出所有的Good Sub-tree，將其分別放到G.1、G.2中。而所謂的Good Sub-tree就是在PE中，

fitness值小於當時threshold值的Sub-tree。通常在找出親代的Good Sub-tree時，threshold值會初使為 0 ，所以所有沒有dead area的Sub-tree都會在這時被淬取出來。

Step.2：把G.1中的所有Good Sub-tree放入Good Sub-tree Pool(GsP)。

Step.3：把G.2中的Good Sub-tree放入GsP，而且放入Gsp的Sub-tree中的block必須尚未放入GsP中。

Step.4：接著把尚未放進GsP中的所有blocks，嘗試兩兩彼此做水平或結合。當結合出來Sub-tree的fitness小於等於當時的threshold值，就把這個Sub-tree丟進GsP裡。這個步驟一直重覆，持續到沒有任何符合threshold值的結合出現為止。

Step.5：若是在Step.4後，仍然有block尚未放進Gsp。就把剩下的blocks跟GsP中的所有Sub-trees，兩兩的嘗試做垂直和水平的結合。一但有Good Sub-tree產生，便放入GsP中。這步驟一直重覆到沒有任何可能的結合發生為止。

Step.6：把GsP中的所有Good Sub-tree，嘗試兩兩做水平或垂直的結合。有新的Good Sub-tree產生，便放回GsP取代掉

原來的Good Sub-tree。這步驟一直重覆到沒有任何可能的結合發生為止。

Step.7：當Step.4、5和6完成之後，還有可能會有尚未放進GsP中的block，而GsP中也還有未結合在一起的數個Good Sub-tree。這時便將threshold稍微加大，允許在Good Sub-tree中有些許的dead area存在。然後重覆Step.4、5和6。一直到所有的block都已放進GsP，而且所有的Good Sub-tree也都已經完全結合在一起，成為一個完整的Floorplan便結束步驟。

Step.8：把最後產生的結果，經過Fitness function，計算出它的fitness值。當它的fitness比親代的還要好，便把比較差的親代給取代掉。

在Mutation方面，GFA有四種不同的動作，而每次執行Mutation的時候，會隨機選擇一種動作進行。藉著隨機所做的動作，來擴大solution space使其跳脫local optimum。而這四個動作分別為：

Op.1：complement a chain

Op 2；exchange a sub-tree and a leaf

Op 3；cut a leaf and insert it at a random position

Op 4；cut a sub-tree and insert it at a random position

第四章 作法

在Floorplan 中數個block為水平或垂直abutment，這些block就必須以規定的方式作結合在一起。根據規則整理推導出一個表格。如Fig.20, 左邊fl為左子樹, 上面fr為右子樹. F表示沒有任何的abutment constraint， V-abu(V)和H-abut(H)各表示vertical和horizontal abutment。其他的就是結合的方式，V表示垂直結合，H水平結合，F表示沒有限制結合的方向，N就是不結合。

f_l \ f_r	(F)	V-abut (V)	H-abut (H)
(F)	F	N	N
V-abut (V)	H	V	N
H-abut (V)	V	N	H

Fig.20 Merge Rule

當兩個block或sub-tree同時為V-abut或H-abut的屬性時，就做垂直或水平合併。而若是其中一個為F，沒有abutment constraints就依另一個的屬性是V-abut還是H-abut，作垂直或水平結合。但是沒有abutment constraint的block 或sub-tree一定要在左子樹。這是為了避免因先前的結合而使得接下來的結合不符合規則。如Fig.21，左邊的結果就不合constraint的規則。當兩個交換結合換成右邊的形式之

後，再下來的結合就能符合constraint的定義。

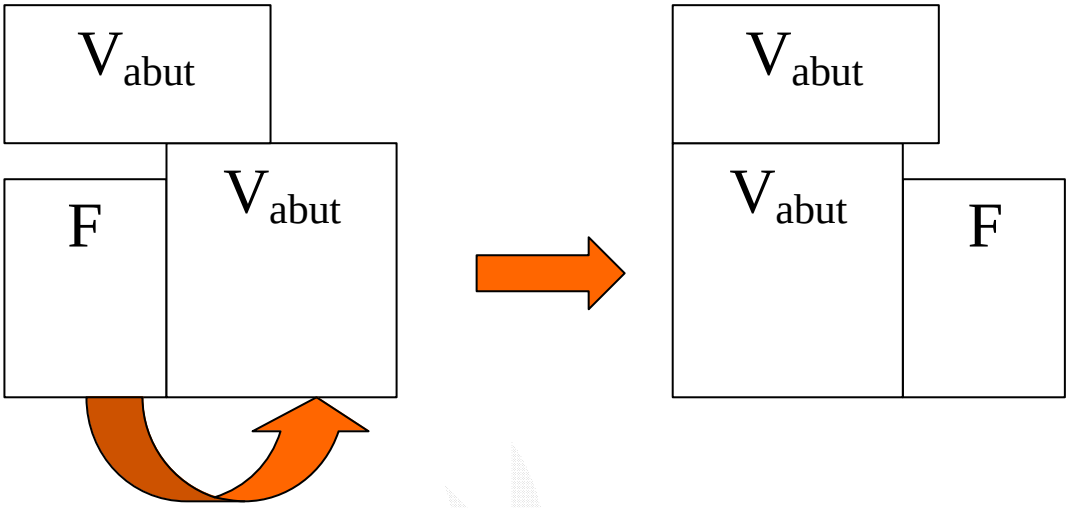

Fig.21 Swap to merge

由於GFA是bottom-up的形式，所以看不到global view 。因此當兩個block或sub-tree分別是V-abut和H-abut的屬性時，因為不知道兩個結合之後接下來遇到的哪一種的constraint，所以便不予以結合，=再另外找其他的結合。當一個sub-tree裡面，所有擁有同性質的abutment 都已經結合在一起時，這個sub-tree便成為沒有abutment constraint的sub-tree。

根據上面的規則定出下面在merge時的檢查流程：

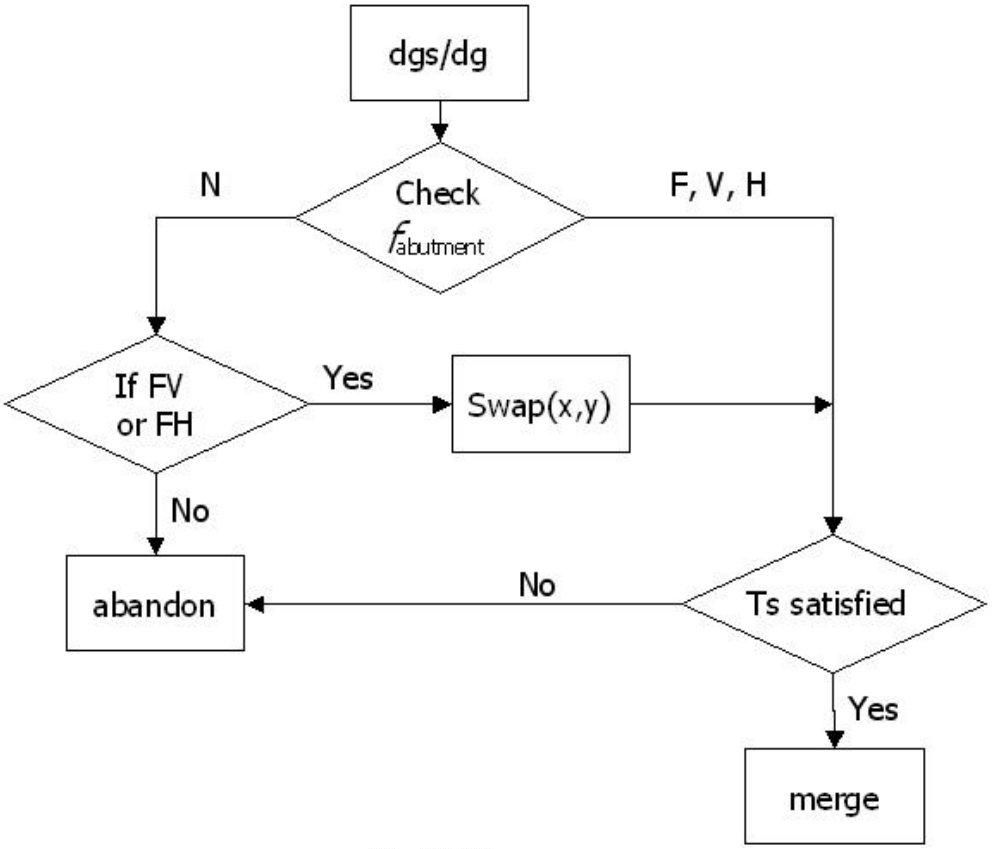

Fig.22 Merge Flow

其中$f_{abutment}$的判斷依照下列的條件判斷如下：

$$f_{Abut}(x,y)=\begin{cases}F, & if\ (x,y)=(F,F)\\ V, & if\ (x,y)=(H,F)\ or\ (V,V)\\ H, & if\ (x,y)=(V,F)\ or\ (H,H)\\ N, & otherwise\end{cases}$$

由結合的規則觀察，有constraint的block一定在左子樹，推斷出有constraint的sub-tree，其PE的第一個block的constraint定義為sub-tree的constraint。然後在去搜尋sub-tree中是否已經有所有相

同constraint的block，如果是的話就把屬性改為F，反之則不變。

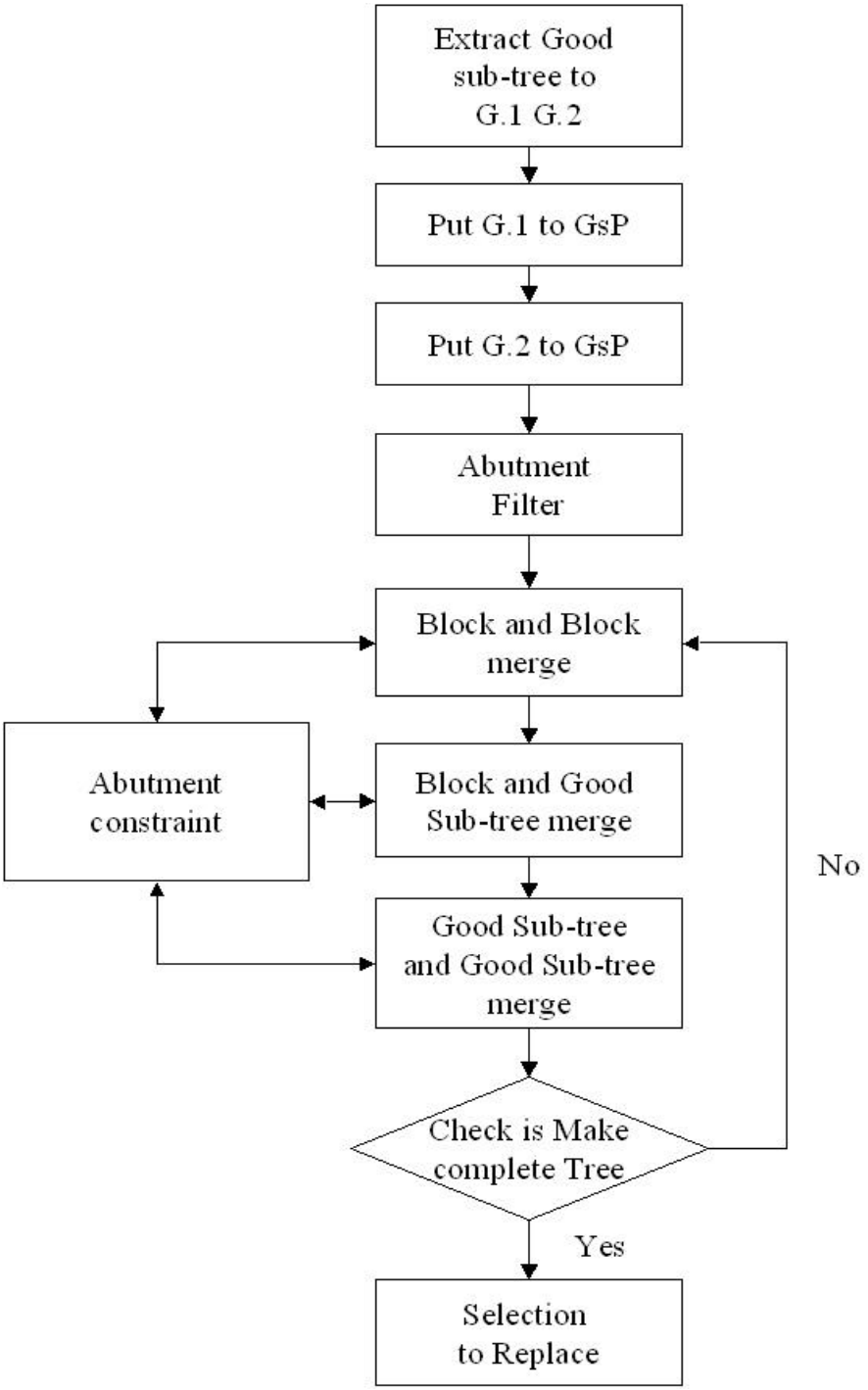

Fig.23 GFA crossover with Abutment constraint

定完結合的方式跟判斷之後，接著便把它加進原先的GFA裡。如

Fig.23，在原先crossover把parent的優良基因extract進GsP之後，跟開始作Block跟block結合之間。多加個check abutment constraint，檢查放進GsP中的Good sub-tree是某符合Abutment constraint，如果符合就保留，不符合的話就從GsP中移除。再把經過篩選的GsP，接著繼續做block的merge。在每一次作block & block、block & sub-tree、sub-tree & sub-tree merge的時候都必須要經過merge rule的判斷、以及符合當時的Ts值，才能進行結合。

檢查GsP的Good sub-tree是否符合constraint時，把PE由左到右掃過一遍。當讀到的是block就把它的constraint的屬性丟進stack中，當遇到結合的符號，在依序由stack取出，由結合的方式判斷是否符合規則。接著判斷這個子樹是否有全部同族群的abutment constraint。把判斷後的屬性再? 回stack中，等到全部掃完一遍就知道Good-sub tree的constraint紀錄下來，若是不符合merge rule在找到違法的時候就會停止，不會在繼續檢查下去。

block & sub-tree和sub-tree & sub-tree merge時，進行rotate可能使得結合後的fitness符合當時的Ts值。但是由於有constraint的sub-tree一但經過rotate 再進行結合，可能會變成違反定義的情形出現。所以在merge時除了檢查constraint決定是否結合之外，也要依照是否為sub-tree而且有無constraint來決定是否rotate。

第五章 測試結果

這個程式全部都是用C++寫的，所有的測試數據是在P-3 800的機器上面跑的。所有的block都是soft block，aspect ratios 在 0.2和5之間。拿ami33、ami49以及n100的benchmarks 針對不同數目的constraints來做測試，結果如Table.1。

Result of abutment constraint testing				
	Constraints number	Dead Area(%)	Area (mm^2)	Time (sec)
ami33	10	0	1.156	57.636
	12	0	1.157	70.268
	14	0	1.157	88.907
ami49	10	0.2	35.532	107.991
	12	0.4	35.613	129.769
	14	0	35.473	145.489
n100	10	0.7	0.181	357.278
	20	1.5	0.182	521.265

Table.1：測試結果

所有求出來的解，都符合abutment constraint的定義。所有的面積也都幾乎沒有dead area。所花時間隨著constraint的數目增加而上升。下面是幾個結果的圖。

Fig.24 Ami33 **的結果，**block 5,6,7,8,9 **為水平** abut。block 23,24,25,26,27 **為垂直** abut。

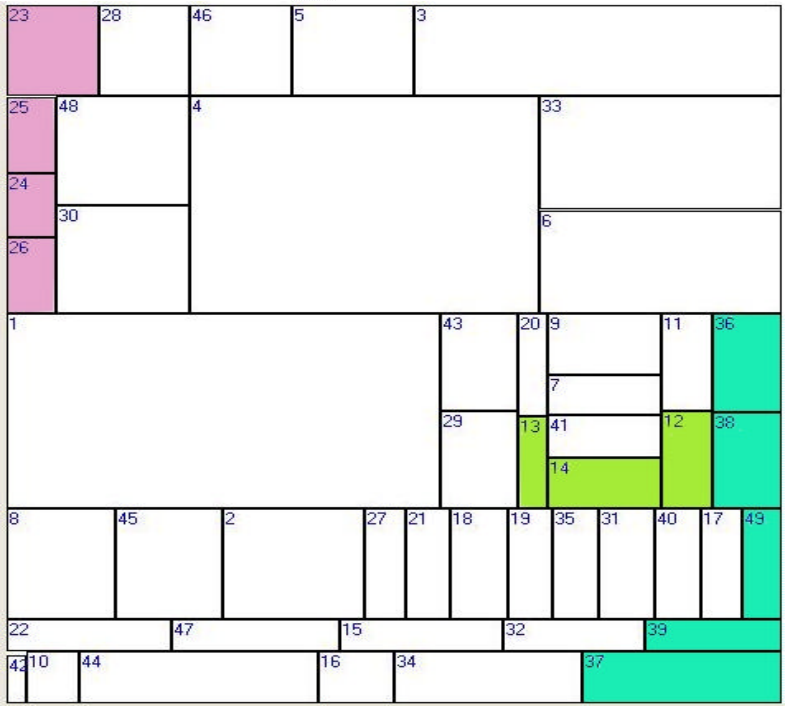

Fig.25 Ami49 **的結果，**block 12,13,14 **為水平** abut。block 23,24,25,26；block 36,37,38,39,49 **為垂直** abut。

Compare of abutment constraint testing					
		CBL		Ours	
	Constraints number	Dead Area(%)	Time (sec)	Dead Area(%)	Time (sec)
ami33	4	2.99	86.4	0	23.01
	6	2.57	90.5	0	30.61
	8	2.58	92.3	0	32.42
ami49	7	3.21	129.3	0.04	61.76
	8	2.44	148.5	0.03	65.42

Table.2：與 CBL 結果之比較

上列數據是將我們的結果拿來和CBL的結果數據[2]相比較。雖然定義constraint的block不一定一樣，但在同樣數目的constraint和相同的soft aspect ratios下，時間跟結果都相對有比較好的結果。

第六章 結論

以整各結果來看，用GFA來解決Abutment constraint的問題相當的合適。所有實驗結果，都能夠符合Constraint。而且善用slicing floorplan的優點，使得所有解的Dead area都趨近於零。

GFA可以由子代繼承親代的優良基因，在Good Sub-tree的聚集下得以快速收斂得到好的解，這點也反映在求解的時間相對的比較快，可以在比較短的時間內得到比較好的解。

和別人的數據相比較，雖然我們的方法的solution space比較小，只能找到slicing 的floorplan 但是卻在時間上得到了好處。而解也沒有比較差，甚至更好。由此可見，使用GFA來解abutment constraint非常的成功。

接下來，可以以此繼續解決L/T Shape以及Alignment constraint，更甚至可以擴展到解決boundary constraint。相信都可以得到不錯的解。

參考資料

[1] F.Y. Young, M.D.E. Wong, and Hannah H. Yang, "On Extending Slicing Floorplan to Handle L/T-Shaped Modules and Abutment Constraints," Proc.WCC '2000,2000,pp.269-276

[2] Yuchum Ma, Xianlong Hong, Scheqin Dong, Yici Cai, Chung-Kuan Cheng, Jun Gu, "Floorplanning with abutment constraints based on corner block list," INTEGRATION VLSI jouenal 31 (2001),pp.65-77

[3] Evangeline F.Y. Young, Chris C.N. Chu, M.L. Ho, "A Unified Method to Handle Different Kinds of Placement Constraints in Floorplan Design," Proc. VLSID 2002

[4] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu, "B*-Trees: A New Representation for Non-Slicing Floorplans," Proc. DAC 2000

[5] K. Shahoohar and P. Mazumder, "VLSI Cell Placement Techniques," ACM Computing Surveys, Vol. 23, No. 2, June 1991

[6] Yuchun Ma, Xianlong Hong, Sheqin Dong, Yici Cai, Chung-Kuan Cheng, Jun Gu, "Floorplanning with Abutment Constraints and L-shaped/T-shaped Blocks based on Corner Block List" Proc. DAC 2001

[7] H. Murata, K. Fujiyoshi, and M. Kaneko, "VLSI/PCB Placement with Obstacles Based on Sequence Pair," Proc. ISPD, pp. 26–31, 1997.

[8] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-Tree Representation of Non-Slicing Floorplan and Its Applications," Proc.DAC, pp. 268–273, 1999.

[9] M. Kang and W. Dai, "General Floorplanning with Lshaped, T-shaped and Soft Blocks Based on Bounded slicing Grid Structure," Proc. of ASP-DAC '97, pp.265-270, 1997.