CS 5160-00 FPGA Arch. & CAD

Final Project

109062637 藍雍淵

---

0. How to compile and execute program:

Compile: go to /src directory and type the command “make”
Execute: go to /bin directory and type the command which follows this format:

```
./<exe> <info file> <nets file> <placement file>
```

e.g.:

```
./fpga_project ../benchmarks/frisc_4.info ../benchmarks/frisc_4.nets ../outputs/frisc_4.placement
```

1. Implementation:

這次我的作法算是 random-based 的實作方式，以下先說明主要會用到的 maintain 的訊息。

initial information: 每個 module(LUT/FF)會記錄放在哪一個 CLB 座標，這個座標以 pair(x, y)紀錄，每個 CLB 也會 maintain 目前放了幾個 LUT/FF 的資訊，以 pair(# of LUT/# of FF)紀錄。

以下主要分為兩個步驟：

(1) initial placement: 以 LUT/FF 的編號依序安排這些 module 的 CLB 初始位置，random 產生兩個數字(x, y)，檢查座標為(x, y)的 CLB 目前擁有的 LUT/FF 數量，如果小於 2（還有空間放），則將此 module 放入座標(x, y)的 CLB，否則重新 random 產出一個新的位置嘗試放入。重複此過程直到所有 module 都被放好。

(2) placement: 在此支援兩個動作，分別是 move 和 swap，以 greedy 的方式進行，也就是當作完動作後 HPWL 有比原本更好才會接受此次動作。以下分別說明兩個動作。

   a. move: random 選一個 LUT/FF，隨機選取一個目前 LUT/FF 還有空位的 CLB 位置，嘗試放入並檢視 HPWL 是否會變好（小於等於原先的線長），如果會變好則接受，否則將此 LUT/FF 放回原先的位置。

   b. swap: random 選取兩個 LUT/FF （兩者皆為 LUT 或皆為 FF），將

此兩個 module 的位置交換（換到對方在的 CLB 位置），並檢視 HPWL 是否會變好（小於等於原先的線長），如果會變好則接受，否則將兩者放回原先的位置。

2. Discuss and Analyze the Problem/Algorithm:

在此說明這次作業會選擇這種 greedy and random algorithm 的理由：

(1) 先檢視了此次作業的 testcase，發現這些 design 的規模都滿小的，加上 FPGA 上 LUT/FF 只能放在 CLB 位置這種「一個蘿蔔一個坑」的特性，導致 solution space 基本上並不大，而且 time limit 又給了 30 分鐘，讓我覺得用 random 不斷的嘗試應該能讓最後的 HPWL 結果不差。（其實相當於窮舉，因為 move 的動作也有機會讓任何 module 到任意位置，所以邏輯上 solution space 的任何解都有可能走到，只是不一定剛好走的到）。

(2) 在有了 (1) 的思路之後，做過兩種版本，一種是只支援 swap，另一種是 swap 和 move 兩種動作都支援（並且沒有分配兩個動作頻率的比率，直接 random 決定要做哪一個動作），結果發現兩種動作都支援的演算法結果會比只支援 swap 動作的結果要好上不少。猜測原因是 random initial placement 甚至可能會使某些 CLB 沒有任何 module 放置，而 move 這個動作可以向外尋找其他更好的位置來擺放 module，但只支援 swap 時會讓 initial placement 的結果限縮 solution space。因此最後就選擇了兩個動作都支援的版本。

(3) 後來做了一些時間上的測試，讓我發現隨機演算法在這次作業的 testcase 上的 wirelength 會在起初的幾分鐘就急速地下降。

以 alu4_4 舉例，初始 wirelength 為 45905，執行 28 分鐘後的 wirelength 為 9187.0，但其實只做 6 分鐘就能做到 wirelength 9721.0 的結果。

但不確定後來下降速度趨緩是因為 testcase 太小已經趨近很好的結果，還是這是 random based 演算法的問題。

(4) 因為有了 (3) 的疑惑，所以曾經嘗試將程式改寫成 simulated annealing 的版本，但後來發現因為都是 random-based 的演算法，但 SA 的隨機和不確定性又更高，再加上這次作業 CLB 位置固定、原本的方法理論上就可以走過任何 solution space 上的點，而且在嘗試過後發現 performance 沒有明顯比較好（或許是參數調整吧），卻會導致整個程式很難定義出一個能試用在各個 case，很 general 的參數設定（SA 的溫度 T、溫度下降比

率α、內圈執行次數等），所以最後就繼續使用 greedy random algo. 而沒有選擇 SA。

(5) 後來認為其實可以考慮使用 SA + greedy random 的做法，就是當溫度過低過後，如果還有時間就繼續做原本的 greedy random algo. 直到時間接近 30 分鐘的上限為止，不過最後來不及嘗試和做出一個更好的結果。

3. Wirelength and Run Time:

在此段落測試環境為自己 lab 的 Server。

CPU: Intel i7-8086K CPU @ 4.00GHz

Ram: 64GB

Run time：以 C++ 的 <chrono> header 來作為程式執行的 timer。

	tseng_4	alu4_4	diffeq_4	frisc_4	s38417_4	clma_4
wirelength	8433.75	9187.0	11692.0	48796.0	113518.0	150207.5
run time(sec.)	1680	1680	1680	1680	1680	1680
run time(min.)	28	28	28	28	28	28

4. Experiences & Conclusion

這次作業因為沒有歷年的數字參考，所以很難評價自己最後 performance 的好壞，但後來猜測如果用 analytical 或 partition-based 的方式搞不好 performance 會吊打我這種 random-based 的方法。另外，因為寫過王廷基老師 Physical Design Automation 課程的 floorplan/placemen 的作業，在 floorplan 作業時發現 greedy random 的作法會和 SA 有一段 performance 上的差距，但在這次的作業卻還好，我想或許是 FPGA 在擺放位置上比較固定的因素導致的，不太會發生做 fixed-outline floorplan 時，搬動一個 module 會造成其他 module 很大影響的情況。

（下為移動一個 module 會影響其他 module 許多的示意圖）