大家好，我們是李兆翔與李子杰，我們做的題目是 ProblemB。

這是我們的 Agenda，大概會分成介紹、資料結構、演算法與結果。

Introduction:

Problem B 要求我們的是開發一個 Banking 或 Debanking 的演算法，讓以下的指標能夠達到最佳，包含最佳化 PPA、最小化 Bin 的密度超過限制的數量、以及最大化 TNS，即負的 Slack 加總。

Ideation process:

我們解決問題的思路是這樣子的：首先參考 PA 的寫法寫出一個 parser，再將資料存到我們設計的資料結構中，再透過演算法把問題解決。

Parser 寫完後，原本想使用 CPLEX，把整個 Problem B formulate 成一個 ILP，但嘗試後不可行，因而轉為 PA2 中曾經使用的模擬退火演算法，其中的過程後面會提及。

Parser:

首先介紹 Parser，在 Sample Input 中，有像左邊這些全域參數，我們的做法就是將他們存成全域變數，如 Die 的大小、cost 的權重...等。

再來是 Input data 中的元件部分，我們是使用 unordered map 取代 vector 存取元件資料，這樣可以直接用類似於 Array 的方式直接存取指定資料，而不用迭代器搜尋，是一種讓整體效率提升的寫法。 如右下角的例子，我們要存取 FF

cell library 的 Gate power，就可以直接用中括號取值。

接下來我們介紹資料結構。 首先是 Pin 資料結構，是用來存 Input、Output 與 FF 接的 Pin，內部資料有 Pin 腳的名字、x 軸與 y 軸。

再來是 Flipflop，我們分成兩部分，一部分是 Cell library 中的 flip flop，如右圖，是存種類的相關資訊，如種類名稱、幾 bit、長寬、pin 數、Qpindelay、GatePower、與 Pin 腳資訊；而另一種是 Instance flipflop，相較前者，除了用指標存這個 instance 的種類外，還另外存了名字、x,y、Pin 腳的 Timing slack 與接到的 clk 種類，方便在之後 Banking 或 Debanking 時判斷。

而 Gate 與前述的 Flipflop 類似，一樣分成 Cell library 的 Gate（存種類）與 Instance 的 Gate（存 x,y 與對應種類）。

而再來是 Net 與 Placement Rows，Net 的部分是紀錄 Pin 腳的連線方式，由於有其順序，所以使用 vector 而非 unordered map；而 Placement rows 是在擺置元件時要對齊的格子。

Cplex:

我們一開始會選擇 Cplex 的原因是因為 Problem B 有明確的 Objective function，還有題目本身的 constraints 也都有較清楚的解釋，再加上我們剛寫完 PA3，感覺可以嘗試應用。

但後來嘗試過後發現建立 constraints 時遇到最主要的難點是在於 Banking 與 Debanking 的邏輯判斷，以及如何設定 notation，及其對應的 constraints，

如上述提到的 banking 邏輯，其中一個 constraint 就是只能合併同 clk 的 FF，再來如正確的接線(map)、元件擺置要符合 placement rows，且不能重疊...等等的許多限制，我們後來並沒有完成用 ILP 解決問題。

SA:

而最後我們是使用模擬退火法生成一組解，首先，cost 的定義與前面的 objective 一樣，都是最小化不好的效能指標，TNS 的計算是由 Expression 回推到 Instance 以更新每個 Pin 的 timing slack，最後再把總共的負 slack 傳回 cost。而 Area、Power 只需要迭代、加總即可。至於 Bin density 則需計算在元件周圍的 Bin 的佔用面積%數，如果超過則加 1，直到迭代完所有 Flipflop。

Expression 的靈感則是來源於 PA2 的 slicing tree，我們建立一個 vector<string>，讓裡面只包含 Instance 與 type，讓它能更容易做 perturbation。

也跟 Slicing tree 類似，我們也要檢查這個 expression 是否是合理的，所以我們每次 perturbation 完都會執行這個 Is_valid 程式去判斷這個 perturbation 是否合理，有點像 balloting property。

最後則是 Perturbation。我們的想法是：隨機選擇 " 一種 " Flipflop，然後再從 Instance 中尋找能夠 Bank 或 Debank 的 instance。

以下面這個表達式為例，這個的意思是 reg1 與 reg2 是合併成”SVT_FF_2”

這種類型的 Flipflop，而 reg3 是 SVT_FF_1 這種類型的 Flipflop，以此類推。

如果對這個表達式做 Perturbation，則先隨機選擇一種 Flipflop。假如這裡選到的是 SVT_FF_2 這種 flipflop，2bit，那 instance 則會選到目前只有 1-bit 的 reg3 與 reg4，確認 clk 來源相同即可 bank 成一個 2-bit Flipflop，就會變成下面這樣。

Results:

這就是我們的執行結果，我們的演算法雖然仍有一些問題，如 Sample case 結果並沒有用到 Banking，可能是 Cost 的計算仍有一些問題，但我們有使用官方提供的 sanity 驗證 output 檔案是對的，這就是我們目前的進度，以下是分工表，我們報告到此結束，謝謝各位。