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。