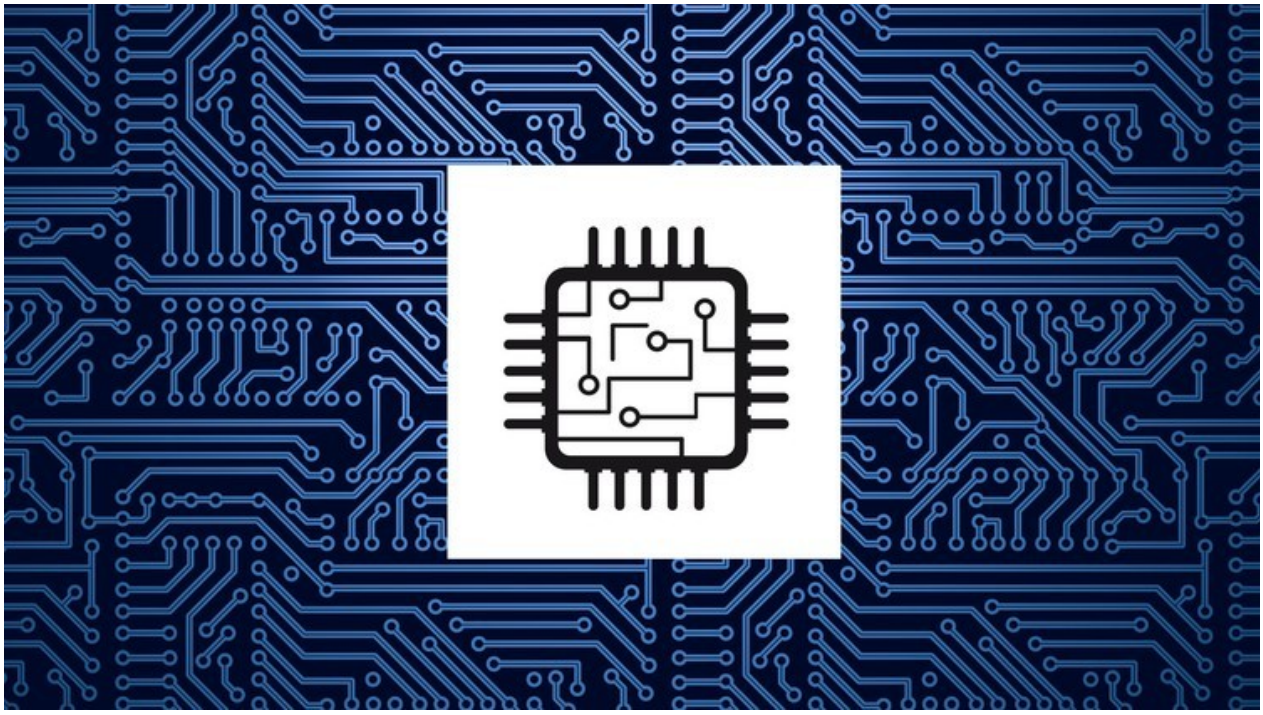
PROJET : ViaMIN

BINOMES: NAIT SLIMANI kamel & SALHI Ramdane



Intérvenants : M. Pierre Fouilhoux
M.Marvin Lasserre

2I006 2018/2019

PARTIE VIA Minimisation

➢ Partie A :

Parmi les problèmes les plus courants dans le domaine de l'électronique est on trouve le problème de conception de circuits VLSI qui est le processus de positionnement des circuits VLSI sur des supports en prenant compte toutes les caractéristiques des réseaux.
Ce processus se découpe en plusieurs étapes et on distingue trois principalement trois qui sont le placement des composant , le routage des réseaux et l'affectation des réseaux au faces .
Cette dernière consiste a trouver les intersections entre les réseaux a vrai dire entre les segments des différents réseaux et cela peut être d'une complexité polynomiale avec des algorithmes naïfs, donc cette partie A de ce projet consiste a implémenter des algorithmes plus performant pour pouvoir traiter des très grandes instances ou cet algorithme naïf peut être très long et pour cela on a un algorithme qui est une sorte de balayage avec une ligne verticale sur les segments horizontaux et cela en ayant les segments verticaux qui se trouve dans cette zone de balayage (ou x € [x min ,x max] .
Et cela nous diminue la complexité sauf dans des cas rares d'instance on pourra avoir une complexité polynomiale en $O(n^2)$ si la liste des segments qui sont dans la liste actuel est très grand qui vas nous augmenter la complexité de l'insertion et de la suppression au pire des cas O((n-1)*n/2), par exemple si on a n segments horizontales et superposés les uns au dessus des autres qui fera qu'on rajoute a chaque fois a la fin de la liste et on supprime a la fin de la liste et cela ignore carrément la boucle de recherche des segments horizontaux qui intersecte un segment vertical puisque on a aucun segment vertical .

On a implémenter un troisième algorithme ou on a le même principe de fonctionnement,
 mais on change la structure qui stocke les segment horizontaux qui peuvent intersecter les segments verticaux.
Donc on a choisis de la représenter par un arbre de recherche équilibré AVL.
Et pour la recherche des segments qui peuvent intersecter un segment vertical de v(y1,y2),
on cherche l'arbre ou se trouvent ces segments.
Pour cela on a changé la fonction prem_apres (y) qui nous renvoie un AVL ou se trouve tous les segments qui peuvent intersecter V on cherchant le premiers arbre ou la racine est supérieure a y1 puis dans le résultat on cherche le premier arbre ou la racine est inférieure à y2 donc on aura que les segments inférieurs a y2 mais pas forcement supérieures a y2 .
La complexité sera O(n*log(n) .

➢ Partie B :

Creation du graphe :

Dans la fonction de l'initialisation du graphe on est partie sur le principe de creation des sommets de tous les points de la netlist puis on cree les sommets des segments on l'a implementé de cette maniere pour poivoir acceder au sommet plus rapidements plusieurs donc le principe est basé sur :
- On parcours sur tous les points de tous les resaux et leur creation

1. -on sauvegarde le debut des indices des points de ce reseau dans le graphe dans un tableau de corespendance entre indices pour recuperer les sommet de points et de segmetns en O(1) car chaque points a au maximum quatre sommets segments adjacents .
2. -On recupere tous les segments de cette netlist avec la fonction déjà definit.
3. -On parcours ce tableau et on cree segments par segments.
4. -On remplis les listes d'adjacence en recuperant les points de ce segments en O (1).
5. -On lit dans le fichier d'intersections les sextuplets un par un .
6. -On recupere les sommets corespendant en utilisant le tableau temporaire qui permet de recuperer un sommet points puis dans la liste d'adjacence du points on cherche le sommet segment corespendant avec le deuxieme point.
7. -On ajoute dans la liste de chaque sommet segment l'autre segment pour avoir le conflit.

Detection de cylces dans le graphe :

elle détecte un cycles a partir d'un sommet (fonction : detectCycleImpair) :
elle meme appelle la fonction cycle chaine qui fait pratiquement tous le travail.
la fonction prend comme aruments :
-un pointeur sur un graphe (Graphe *)
- un tableau de marquage M(int *)(-1,0, 1, 2)
-un entier qui represente le numero neud appelant.
- un entier qui représente le numero du pere du neud appelant
L'algoritheme fonctionne comme suit:
on parcout la list d'adjacence de ce neud et pour chaque fils f on test :
si un Via M[f] =0 on passe au prochain fils
Si non :
si (f n'est pas exploré et difrent du pere du neud appelant)
-On apppele recurssivement sur la fonction Cyclechaine en lui donnant en parametre le le le neud appelant u comme pere
et f a explorer.
On sauvegarde le resultat dans une variable .
Si la variable est non null :
on ajoute le le neud actuel a la liste a renvoyer qui accumule les sommets du cycle en verifiant si on atteind le cycle ou pas encore donc si le debut de la liste est diferent de la fin donc on l'ajoute si non on ajoute plus.
Si non
on test si M[u]=M[f] cas de detection du cycle
on cree une list vide puis on ajoute le sommet f le fils
et le sommet u l'appelant et on met M[u] =-1 pour pouvoir rexplorer si on a un autre cycle qui passe par ce sommet.
Et on retourne la liste et on met le M[u]=-1 pour le rexplorer après.
Si non donc les car restant c'est que on a parcouru toute la liste des sommets adajcents sans detecter de cycle donc on met le sommet courant u à 0.

fonction ajout via:
tant qu'il existe des sommets non exploités on iter on recupere le sommet et on lance dessus detect cycle imapaire.

-la fonction ListSomCycle la fonction qui nous forme la list des sommets qui composent un cycle

prend en parametre le Tableau des pere la racine et le neud detectant le cycle v et r comme la racine a ateindre dans le tableau.
- on cree une list vide.
-tanque v est difernt de r on ajoute le sommet corespendant à v dans la liste des sommtes et et on met a jour v a son pere dans le tableau.
-compléxité est O(n) si le cycle contient tous les neud du graphe.
// la complexite de la fonction est a definir

La fonction ajout via on initialise le tableau M des sommets explorés à que des -1 puis tanque il existe un -1 dans ce tableau on relance la detection du cycle sur le sommet .
On initialise le sommet trouvé a 2
On recupére la liste de sommets qui compose ce sommet (detectcycleimpair)
si la liste est vide
alors on chereche le prochain sommet a explorer.
Si non
on cherche le premier sommet points dans la liste on le met à 0 le sommet dans M et S .

La fonction Bicolor elle prend le un graphe et calcul le tableau de via et sommets non via puis tant que il existe un sommet a -1 dans S elle appelle une donction recurssive qui met les tous sommet adjacent a 3-u (1→2) (2→1) et appelle recusivemetns sur les sommets adjacents pour decider de les mettre sur la face 1 ou sur la face 2.