Projet: Le problème de Via Minimization

Ce projet s'intéresse au problème de "Via Minimization" (minimisation du nombre de vias) qui est une étape du processus de conception de circuits électroniques.

Déroulé du projet

L'énoncé de ce projet est divisé en 2 parties :
- Partie A : qui se déroule de la séance 4 à la séance 7 avec rendu lors de la séance 8 (semaine du 9 mars). Il s'agit d'un rendu intermédiaire comportant votre code et un rapport très synthétique d'analyse de performance.
- Partie B : qui se déroule de la séance 8 à 10 avec rendu final lors de la séance 11 (semaine du 13 avril). Il s'agit du rendu final de l'ensemble du projet et comportant un rapport complet reprenant le rapport de la partie A.

Séance 11 (semaine du 13 avril) :
Ce projet est à rendre lors de la semaine 11 lors de la séance de TD/TME à votre chargé de TD/TME. A cette occasion, vous lui montrerez votre code, son fonctionnement et ses performances. Vous devez également rendre un rapport décrivant votre code et votre travail de réflexion, ainsi qu'une analyse statistique et théorique de sa qualité. Lors de cette séance 11, un dernier exercice vous permettra de poursuivre le projet et d'avoir des points bonus.

Chaque partie est subdivisée en exercices qui vont vous permettre de concevoir progressivement le programme final. Il est conseillé de suivre les étapes données par ces différents exercices. Chaque exercice contient des notions qui seront introduites en cours et en TD en parallèle. Il est impératif que vous travailliez régulièrement afin de ne pas prendre de retard : chaque exercice correspond ainsi à une ou deux séances de TME précises où les chargés de TME peuvent vous aider sur l'exercice en cours.

Attention : cette énoncé peut connaître des petites évolutions ou un apporter des précision : consulter fréquemment la page du module :
https ://www.licence.info.upmc.fr/lmd/licence/2014/ue/2I006-2015fev/

Définitions et cadre du sujet

Cadre du projet : les circuits VLSI

On appelle *circuit VLSI* (Very Large Scale Integrated) un dispositif électronique permettant la réalisation d'une fonction logique connue (stockage mémoire, calcul numérique, pilotage de robot,...). Un circuit est entièrement décrit par un schéma électronique donnant les liens logiques entre les différents éléments du circuit. Dans ce projet, nous considérons des circuits dont on ignore la fonction et l'utilisation. Cette simplification permet ainsi de donner une définition simple d'un circuit.

Notons qu'il existe principalement deux types de technologies : les cartes de circuits imprimés (Printed Circuit Board ou PCB) et les circuits intégrés.
Les cartes de circuits imprimés servent en général à rassembler les gros composants électroniques de manière à former par exemple les cartes mères des micro-ordinateurs, les systèmes de pilotage de robots ou les appareils de mesure. Cette technologie consiste à souder les composants sur la face supérieure et/ou inférieure d'une carte en époxy dans laquelle circulent des pistes de cuivre correspondant aux réseaux. Parmi ces pistes de cuivre, certaines permettent de véhiculer l'alimentation des composants.
Les circuits intégrés nécessitent une technologie plus récente qui a permis la miniaturisation des processeurs, ainsi appelés *puce* (chip, en anglais). La plupart des circuits intégrés sont créés sur un support en silicium dans lequel sont gravés ou superposés des pistes et des composants. Cette technologie permet de concevoir des circuits de tailles et de complexités si importantes que certains observateurs les présentent comme les objets les plus complexes créés par l'homme. Cela induit pour leur conception la nécessité d'utiliser des procédés entièrement automatisés et optimisés pour leur conception et leur fabrication.

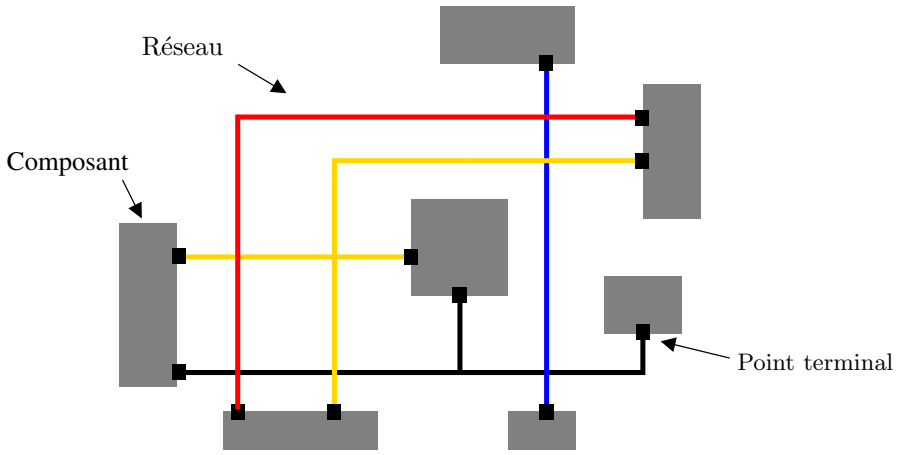

FIGURE 1 – Un exemple de circuit

Dans ce projet, *circuit* sera simplement défini par un ensemble de *composants*, un ensembl de *points terminaux* et un ensemble de *réseaux* :

- Les composants seront considérés comme des boîtes noires pouvant schématiser les composants de base (transistors, condensateurs,...) ou des macro-composants (processeurs, sous-circuits,...).
- A chaque composant correspond un ensemble de *points terminaux*. Un point terminal est une zone (assimilé à un point) du composant qui connecte le composant au reste du circuit.
- Un *réseau* est un ensemble de points terminaux reliés électriquement par des "pistes" conductrices : c'est-à-dire un ensemble de lignes droites ou courbes reliant plusieurs composants.

La figure 1 illustre les définitions précédentes. Les composants (donnés par les boîtes grises) sont reliés par quatre réseaux. Chacun des 4 réseaux est donné par des lignes de couleurs différentes. Certains réseaux ne sont que des lignes droites mais certains se ramifient en plusieurs segments de droite. Par exemple, le réseau rouge est constitué de 2 segements de droites reliés au même point en formant un coude. Le réseau jaune comporte 5 segments de lignes droites dont 4 d'entre eux sont reliés au même point.

Les étapes de la conception de circuits VLSI

On appelle *problème de conception de circuits VLSI* (VLSI design problem, en anglais) l'ensemble du processus qui permet de positionner le circuit VLSI à sur un support, en prenant en compte toutes les caractéristiques de la technologie choisie concernant les composants et les réseaux.
Le problème de conception de circuits VLSI se découpe traditionnellement en plusieurs étapes. En effet, ce processus est très complexe et il serait difficile de tout concevoir en une seule étape. De plus, cette division permet à l'opérateur qui le conçoit d'intervenir manuellement au cours du développement pour orienter la conception. On distingue principalement trois étapes : le placement des composants, le routage des réseaux et l'affectation des réseaux aux faces :

- *L'étape de placement* (placement, en anglais) consiste à positionner les composants sur le support. Cette étape est réalisée en essayant de disposer les composants au mieux les uns à côté des autres en fonction du fait qu'ils doivent être connectés entre eux.
- Etant donnés les coordonnées des points terminaux des composants et la connaissance de ceux qui doivent être reliés par un réseau, *l'étape de routage* (routing, en anglais) consiste à donner des emplacements pour des pistes reliant tous les points terminaux d'un même réseau. En fait, la figure 1 indique un circuit à l'issue de l'étape de routage.
- L'étape *d'affectation des réseaux aux faces* (layer assignment en anglais) consiste à éviter de faire chevaucher deux pistes de réseaux différents. En effet, chaque réseau véhicule une information qui lui est propre et ne doit pas être connecté à un autre réseau. Heureusement, le support d'un circuit VLSI possèdent une face supérieure et une face inférieure. En fait, il est souvent impossible de faire circuler tous les réseaux d'un circuit sur une seule surface sans que ceux-ci ne se chevauchent. Les points terminaux des composants sont accessibles sur les faces (en traversant le support), mais un segment de pistes n'est présent que sur une des faces.

Problème de Via Minimization

Dans ce projet, nous allons nous intéresser à cette dernière étape de conception appelée "affectation des réseaux sur deux faces".

Lorsque deux segments de pistes appartenant à un même réseau sont sur deux facess différentes, le réseau doit néanmoins rester connecté. Concrètement, un changement de face est rendu possible par le placement d'un *via* qui permet de connecter électriquement les différents segments de piste d'un réseau entre des faces différentes. Pour un circuit imprimé, un via est un trou gainé de cuivre, percé dans le support en époxy. En circuit intégré, un via est un point de contact entre les pistes de deux couches adjacentes.

La figure 2 a) présente un circuit logique comportant trois réseaux reliant chacun deux points terminaux. Ce circuit ne peut être dessiné sur une seule couche sans que deux pistes ne se chevauchent. Par contre, sur deux faces, une solution est rendue possible par le percement d'un via. La figure 2 b) donne une solution possible sur deux faces : chaque face est alors indiquée par un type de traits différents (A : trait plein et B : train pointillé). On peut remarquer que l'un des réseaux est découpé en deux segments qui sont affectés sur deux couches faces. Un via percé entre les deux faces permet de relier les deux sous-réseaux.

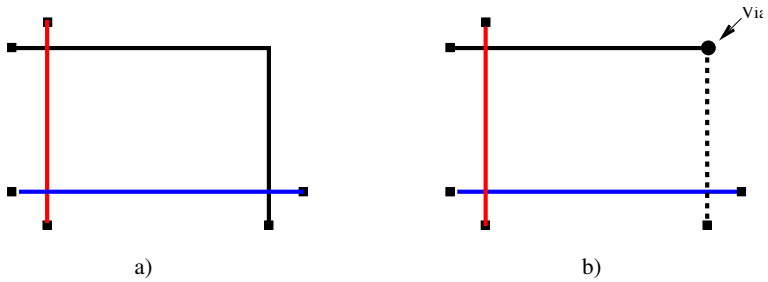

FIGURE 2 – a) Un exemple de circuit et b) son dessin sur deux couches avec un via

Comme on peut le voir sur cet exemple, les vias sont utiles et fréquemment nécessaires, mais ils dégradent la performance du circuit et peuvent être la cause de contraintes électrostatiques. De plus, les vias ont un coût de production non négligeable. On désire donc placer un nombre minimum de vias.

On considère dans ce projet le problème consistant à répartir les segments de pistes des réseaux d'un circuit VLSI entre deux faces, sans que deux segments appartenant à des réseaux différents ne soient connectés. On désire trouver une telle affectation en plaçant un nombre minimum de vias : c'est *le problème de Via Minimization.*

PARTIE A : Circuits VLSI et recherche d'intersections

Cette première partie a pour objectif de lire des instances du problème de via mnimization et de rechercher les intersections éventuelles entre segments de réseaux différents.

Exercice 1 – Manipulation d'une instance du problème (séance 4)

On désire tout d'abord manipuler et afficher une instance du problème de Via Minimization. Un telle instance est aussi appelée une *netlist* c'est-à-dire la liste des réseaux.

Comme les pistes ont déjà été dessinées sur un plan, un *réseau* est un ensemble de points reliés par des *segments* de pistes. On appellera ici *segments* une ligne droite délimitée par deux points et on dira que ces deux points sont alors *adjacents*. Comme on l'a vu sur la figure 1, un réseau peut être un simple segment reliant deux points terminaux ou un réseau plus complexe ou des segments sont incidents à un point d'embranchement. On appelera *degré d'un point* le nombre de segments **d'un même réseau** qui part de ce point. Dans toutes les instances rencontrées dans ce projet, les segments seront soit horizontaux (lettre H), soit verticaux (lettre V).

Un instance sera donnée par un fichier d'extension .net correspondant au format donné par l'exemple suivant :

```
3             // Nb de réseau dans l'instance.
0 2 1         // Numero du réseau / Nb de pts du réseau / Nb de segments du réseau
  0 20 40     // Point 0 de coordonnées (20,40)
  1 120 40    // Point 1 de coordonnées (120,40)
  0 1         // segment reliant les points 0 et 1 du reseau
1 3 2
  0 60 0
  1 60 150
  2 20 150
  0 1
  1 2
2 4 3
  0 80 0
  1 80 200
  2 20 200
  3 150 200
  0 1
  1 2
  1 3
```

Dans l'exemple précédent, la netlist de l'instance possède 3 réseaux tels que le réseau 0 est un segment, le réseau 1 est composé de deux segments "en coude", le réseau 3 est composé de trois segments "en T".

Vous trouverez sur le site du module un ensemble d'instances (benchmark en anglais) pour ce projet. Certaines instances ont été générées aléatoirement et d'autres proviennent de circuits VLSI réels.

Pour charger et manipuler en mémoire un réseau, on va utiliser la structure de données suivantes.

```
1  struct segment;
2
3  typedef struct cell_segment{
4    struct segment* seg;
5    struct cell_segment *suiv;
6  } Cell_segment;
7
8  typedef struct segment{
9
10   int NumRes;  /* Numero du reseau auquel appartient ce segment*/
11
12   int p1, p2;  /* Numero des points aux extremites du segment */
13                /* En utilisant la numerotation de T_Pt */
14                /* p1 est le point en bas a gauche par rapport a p2*/
15
16   int HouV;    /* 0 si Horizontal et 1 si Vertical */
17
18   struct cell_segment *Lintersec;  /* Liste des segments en intersection */
19
20 } Segment;
21
22 typedef struct point{
23   double x,y; /* Coordonnees du point */
24
25   int num_res; /* Numero du reseau contenant ce point = Index du tableau T_res*/
26
27   Cell_segment  *Lincid; /* Liste des segments incidents a ce point */
28
29 } Point;
30
31 typedef struct reseau{
32
33   int NumRes; /* Numero du reseau = Indice dans le tableau T_Res */
34
35   int NbPt;  /* Nombre de points de ce reseau */
36
37   Point* *T_Pt; /* Tableau de pointeurs sur chaque point de ce reseau */
38
39 } Reseau;
```

En utilisant cette structure, un réseau est donné comme un ensemble indicé de `NbPt` points (numérotés de 0 à `NbPt` -1) dans le tableau `T_pt`. On retrouve les segments en regardant, pour chaque point, les segments incidents à ces points. Pour un segment donné, on donne le numéros de ces points extrémités dans le tableau `T_pt`, son orientation H ou V et la liste de ses intersections avec d'autres segments de réseau différents. **Attention,** cette liste des intersections Lintersec est initialisée comme une liste vide. Nous la remplirons dans les exercices suivants.

Ainsi, la structure de données suivante code une instance netlist complète.

```
1  typedef struct netlist{
2     int NbRes;  /* Nombre de reseaux */
3
4     Reseau* *T_Res; /* Tableau  pointant sur chaque reseau        */
5
```

```
6 } Netlist;
```

On peut remarquer que l'on numérote les `NnRes` réseaux d'une netlist de 0 à `NbRes` -1 et ce numéro correspond à l'index du tableau `T_res`. Ainsi un point du circuit peut être donné par son numéro de réseau suivi de son numéro de point.

Q 1.1 Implémentez un ensemble de méthodes `Netlist` qui permettent de créer et d'allouer une instance de notre structure à partir d'un fichier d'entrée. (Vous pouvez utiliser la bibliothèque d'Entrée/Sortie qui vous a été fournie dans les TME de début de semestre).

Q 1.2 Dans le but de valider votre code de lecture d'instance, construisez une fonction qui affiche dans un fichier le contenu d'une `Netlist` en respectant le même format que celui contenu dans le fichier (cela revient à re-créer le fichier d'entrée). Tester votre code sur plusieurs instances.

Q 1.3 On désirer afficher les instances. Pour cela, on pourrait utiliser par exemple un outil de dessin de segments (gnuplot,...) ou de graphes (graphiz,...). Il est néanmoins difficile de tracer des instances de grandes tailles avec ces outils. Nous allons plutôt utiliser un outil très efficace et très léger : le format postscript. Vous trouverez sur le site du module, une documentation pour utiliser un fichier postscript pour dessiner des points et des segments. Créer un programme `VisuNetlist` qui lit une instance et écrit sur disque un fichier de même nom au format .ps visualisant l'instance.

Exercice 2 – Recherche des intersections (séance 5)

Nous allons dans les deux exercices suivants rechercher toutes les intersections entre segments appartenant à des réseaux différents . A l'issue de cette étape, la structure de données permettra de connaître pour un segment donné la liste des segments qu'il croise (grâce au champs `Lintersec`) afin de pouvoir placer ce segment sur une face ou l'auttre du support du circuit.
Les intersections dans nos instances n'ont lieu qu'entre un segment horizontal et un segment vertical appartenant à des réseaux différents. Ainsi, nous pouvons vérifier simplement si deux segments s_v et s_h s'intersectent : il suffit de vérifier si l'abscisse du segment vertical s_v est comprise dans l'intervalle défini par les abscisses du segment horizontal s_h et que l'ordonnée du segment s_h est comprise dans l'intervalle défini par les ordonnées du segment s_v. D'autre part, les coordonnées des points sont des entiers et on suppose que deux points ont même abscisse (respectivement même ordonnée) si leur abscisse (resp. coordonnée) coïncide exactement (ce qui revient à considérer des pistes d'épaisseur inférieur strictement à 0,5).

Q 2.1 Implémentez une fonction `int intersection(Netlist *N, Segment *s1, Segment *s2)` qui vérifie si le segment s_1 intersecte le segment s_2.

La complexité de ce problème de recherche d'intersections a pour paramètres le nomre n de segments de l'instance. Remarquons tout d'abord qu'il existe un algorithme évident en $O(n^2)$ pour cette recherche d'intersections : cet "algorithme naïf" consiste à comparer deux à deux tous les segments.

Q 2.2 Implémenter une fonction `nb_segment` qui retourne le nombre de segments d'une instance netlist.

Q 2.3 Implémenter une fonction retournant un tableau dont chaque case est un pointeur sur l'un des segments. Implémenter la fonction `intersect_naif` pour rechercher ces intersections. Elle consiste à comparer deux à deux tous les segments.

On peut remarquer que dans le cas où tous les segments se coupent les uns les autres, le nombre d'intersections est également de l'ordre de $O(n^2)$: on peut donc en déduire que dans ce pire des cas, cet algorithme naïf est le meilleur possible.

Dans notre cas d'utilisation, pour les instances que l'on manipule, tous les segments sont dans un plan et sont assez courts : ainsi le nombre d'intersections par segment est très limités : on peut l'estimer de l'ordre de $O(n)$. Il est donc intéressant de rechercher un algorithme plus efficace que l'algorithme naïf. Nous allon utiliser une procédure par balayage pour réaliser la même tâche avec une complexité moindre. L'idée est la suivante : on considère une droite de balayage verticale imaginaire qui se déplace à travers l'ensemble de segments, de la gauche vers la droite. A une abscisse x donnée de son balayage, on considère tous les segments horizontaux $T = \{h_1, \ldots, h_{k_v}\}$ coupant la droite et triés par leurs ordonnées. Voir figure 3. Un segment vertical d'abscisse x ne pourra alors interscter que les segments de T.

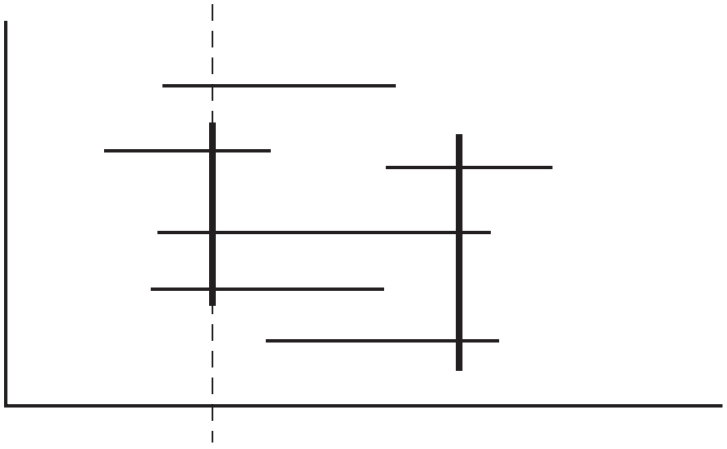

FIGURE 3 – Un exemple de balayage

Pour mettre en œuvre l'idée précédent, on considère l'échéancier des points d'événement qui est une séquence d'abscisses, ordonnée de la gauche vers la droite, qui définit les positions d'arrêt de la droite de balayage. Dans notre cas, les points d'événements seront les extrémités des segments. Il y aura plusieurs comportement lors d'un point d'arrêt :

- si le point d'événement correspond à l'extrémité gauche d'un segment horizontal, il faudra insérer ce segment dans T,
- si le point d'événement correspond à l'extrémité droite d'un segment horizontal, il faudra supprimer ce segment de T,
- si le point d'événement correspond à l'abscisse d'un segment vertical, il faudra générer la liste des segments de T qu'il intersecte.

La structure T qui contient des segments triés selon une clef égale leur ordonnée. On considère les fonctions Insérer(h, T), Supprimer(h, T) et AuDessus(v, T) qui permettre d'insérer un segment v, supprimer un segment v et retourner le segment qui est au-dessus d'un autre segment. On considère

que T peut contenir des segment verticaux dont l'ordonnée sera celui du point le plus bas.
L'algorithme global s'écrit alors comme suit :

```
T ← ∅
Trier les abscisses des extrémités des segments (échéancier)
pour chaque point r de l'échéancier faire
    si r est extrémité gauche d'un segment horizontal h alors
        Insérer(h, T)
    fin
    si r est extrémité droite d'un segment horizontal h alors
        Supprimer(h, T)
    fin
    si r est l'abscisse d'un segment vertical v alors
        Insérer(v, T)
        s_c ← AuDessus(v, T)
        tant que s_c et v s'intersectent faire
            si s_c et v n'appartiennent pas au même réseau alors
                Ajouter s_c à la liste de v
                Ajouter v à la liste de s_c
            fin
            s_c ← AuDessus(s_c, T)
        fin
        Supprimer(v,T)
    fin
fin
```

Algorithme 1 : Recherche des intersections

Q 2.4 On veut utiliser une structure de liste triée pour la structure T. Quelle est sa complexité (pire-cas) ? Si l'on connaît le nombre α du nombre maximum de segments horizontaux traversés par la droite quelque soit l'abscisse du plan, donner une autre mesure de la complexité de l'algorithme ?

Q 2.5 On désire comparer les performance des algorithmes `intersect_naif` et `intersect_balayage`. Pour cela, nous allons tracer les courbes des temps d'exécution des deux algorithmes pour les instances de la benchmark. Est-ce que les courbes théoriques respectent vos estimations de complexité ?