



HAL
open science

Placement et routage de circuits mixtes analogiques-numériques CMOS

Eric Lao

► **To cite this version:**

Eric Lao. Placement et routage de circuits mixtes analogiques-numériques CMOS. Architectures Matérielles [cs.AR]. Sorbonne Université, 2018. Français. NNT : 2018SORUS575 . tel-02924679

HAL Id: tel-02924679

<https://theses.hal.science/tel-02924679>

Submitted on 28 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse de Doctorat de Sorbonne Université

Spécialité Informatique

École Doctorale Informatique, Télécommunications et Électronique

Présentée par

Eric LAO

pour obtenir le grade de

Docteur de Sorbonne Université

Placement et routage de circuits mixtes analogiques-numériques CMOS

Soutenue le 14 Septembre 2018 devant le jury composé de:

Directeur de thèse : **Mme. Marie-Minerve LOUËRAT**

Co-encadrant de thèse : **M. Jean-Paul CHAPUT**

Mme. Sonia BEN DHIA, Professeur, INSA Toulouse,	Rapporteur
M. Philippe COUSSY, Professeur, Université Bretagne Sud,	Rapporteur
M. Luca ALLOATTI, Docteur, ETH Zurich,	Examineur
M. Laurent FESQUET, HDR, Université Grenoble Alpes,	Examineur
M. Hamid KOKABI, Professeur, Sorbonne Université,	Examineur
M. Habib MEHREZ, Professeur, Sorbonne Université,	Examineur
M. Marc SABUT, Ingénieur, STMicroelectronics,	Invité

Sorbonne Université
Laboratoire d'Informatique de Paris 6 - LIP6
Place Jussieu, 75005 Paris, France



Ces travaux de thèse sont sous licence **CC-BY-NC-SA**
<https://creativecommons.org/licenses/by-nc-sa/4.0/>

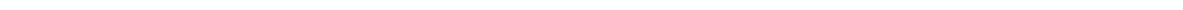
Résumé

Avec l'évolution des procédés technologiques d'intégration, le traitement numérique devient de plus en plus rapide tout en coûtant moins en surface et en consommation d'énergie. La diminution des dimensions est effectuée au détriment de la précision des blocs analogiques. L'idée est de bénéficier des performances offertes par les circuits numériques pour relâcher les spécifications des blocs analogiques et gagner ainsi globalement en surface et consommation. Or les concepteurs de circuits mixtes analogiques-numériques sont confrontés à une situation où ils doivent choisir entre un flot purement analogique et un flot purement numérique, chacun des deux ignorant l'autre.

Cette thèse propose un flot de conception mixte permettant d'unifier le flot de conception numérique et analogique dans la phase de conception du dessin des masques. Le flot de conception se divise entre trois parties majeures : une phase de placement, une phase de routage global et une phase de routage détaillé. Dans la phase de placement, le concepteur est amené à décrire un placement relatif de son circuit sous la forme d'un script permettant à notre outil de générer un ensemble de placements valides respectant les contraintes décrites par le concepteur. Le choix du placement est réalisé de manière interactive à travers une interface graphique permettant de visualiser ce choix. Par la suite, une phase de routage global détermine de manière grossière les chemins les plus courts permettant de joindre les connecteurs de chaque *net*. Ces chemins prennent en compte diverses contraintes du circuit telles que des obstacles ou des contraintes de symétrie. Une phase de routage détaillé vient ensuite compléter la construction et la résolution des problèmes de superposition des fils de routage.

Notre flot de conception est appliqué à des circuits analogiques et mixtes de tailles différentes. Les résultats présentés montrent une capacité à placer et router en un temps court tout en respectant les contraintes des concepteurs. Notre approche a pour objectif de faire appel aux concepteurs et à leurs expériences tout au long de la conception du dessin des masques dans le but de choisir les meilleures solutions de placement routage.

Mots clés : dessin des masques, circuits intégrés, analogique, numérique, mixte, placement, routage global, routage détaillé, CMOS, outils de CAO, automatisation, flot de conception.



Remerciements

Je souhaite exprimer toute ma reconnaissance à Marie-Minerve Louërat et Jean-Paul Chaput, pour avoir tout fait pour rendre cette thèse possible et pour tous les efforts et le temps accordé à l'encadrement de ma thèse. Je vous remercie d'avoir partagé votre expérience et de m'avoir tant fait confiance durant l'intégralité de ces années de thèse mais également durant les années qui ont précédé cette thèse à travers plusieurs stages. Je souhaite aussi remercier Stéphane Fay de m'avoir accordé la chance d'effectuer ces conférences aux Palais de la Découverte durant près de deux années.

Je remercie M. Philippe Coussy et Mme. Sonia Ben Dhia d'avoir accepté d'être membres de mon jury et rapporteurs de mon manuscrit. Je remercie également les autres membres du jury : M. Hamid Kokabi, M. Habib Mehrez, M. Luca Alloatti, M. Laurent Fesquet et M. Marc Sabut de m'accorder de leur temps.

Je souhaite remercier l'ensemble de l'équipe pédagogique du master SESI de m'avoir inspiré durant toutes ces années et d'avoir suscité mon intérêt pour le monde de l'électronique et de l'informatique. Je pense en particulier à Farouk Vallette, Julien Denoulet, Sylvain Feruglio, Dimitri Galayko, Geoffroy Klisnick et Franck Wajsbürt. Je souhaite remercier particulièrement Hassan Aboushady de m'avoir encadré durant mon stage de fin de master et pour les discussions avec son équipe. Je remercie également Haralampos Stratigopoulos, Jacky Porte, Amine Rhouni et Naohiko Shimizu pour avoir pris le temps de partager leurs expériences avec nous.

Merci à tous mes collègues avec qui j'ai passé du temps durant ces années de thèse : Youen Lesparre, Olivier Tsiakaka, Cédric Klikpo, Jad Khatib, Armine Karami, Gabriel Gouvine et Orlando Chuquimia pour tous les moments de joie, tous les conseils et discussions intéressantes qu'on a pu partager. Je voudrais remercier également la gentillesse du personnel administratif en particulier Shahin-Léa Mahmoodian, Sabrina Vacheresse et de toutes les personnes que j'aurais oubliées de mentionner faisant partie du LIP6.

Je pense fort à mes parents et au reste de ma famille pour leurs encouragements et leur soutien dans les moments difficiles. Je remercie tous mes amis qui m'ont tant encouragé et soutenu durant ces années de thèse, leurs encouragements ont contribué à donner le meilleur de moi-même.

Table des matières

Table des matières	vii
Liste des figures	ix
Liste des tableaux	xv
1 Introduction	1
1.1 Contexte	2
1.2 Conception du dessin des masques mixtes actuelle	3
1.3 Enjeux et motivations	5
1.4 Contributions de cette thèse	6
1.5 Plan de la thèse	7
1.6 Références	8
2 État de l'art des outils de CAO pour circuits analogiques et mixtes	9
2.1 Introduction	10
2.2 Les outils de génération de dessins des masques	10
2.3 L'historique des outils de CAO au LIP6	18
2.4 Conclusion	21
2.5 Références	25
3 Placement	29
3.1 Introduction	30
3.2 Formalisation du problème du placement	31
3.3 Les méthodes de résolution existantes	37
3.4 Méthode de résolution	44
3.5 Implémentation du placement	57
3.6 Conclusion	71
3.7 Références	72
4 Routage global	77
4.1 Introduction	78
4.2 Formalisation du problème du routage global	79
4.3 Les méthodes de résolution existantes	86
4.4 Méthode de résolution	89
4.5 Implémentation du routage mixte	103
4.6 Conclusion	127
4.7 Références	128

5	Routage détaillé	129
5.1	Introduction	130
5.2	Achèvement du routage global	131
5.3	Résolution des superpositions	141
5.4	Contraintes des signaux analogiques	148
5.5	Conclusion	152
5.6	Références	152
6	Résultats	153
6.1	Introduction	154
6.2	Amplificateur à transconductance de type Miller	155
6.3	Amplificateur à source de courant ajustable	162
6.4	Transconductance différentielle configurable	166
6.5	Conclusion	169
6.6	Références	169
7	Conclusion et perspectives	171
7.1	Conclusions	172
8	Publications	177
8.1	Références	177
A	Annexes	I
A.1	Script de l'amplificateur à source de courant ajustable	I
A.2	Script de la transconductance différentielle contrôlable	IV

Liste des figures

1.1	Croissance des principales catégories de marché de circuits intégrés[1] . . .	2
1.2	(a) Flot de conception du dessin des masques numérique (b) Flot de conception du dessin des masques analogique	3
1.3	Flot de conception mixte : unification du flot de conception numérique analogique	6
2.1	Comparaison des approches des outils de génération de dessins des masques pour circuits analogiques et mixtes	12
2.2	Structure et flot de conception d'ALSYN[6]	13
2.3	Flot du générateur de dessins des masques de BAG[7]	13
2.4	Flot de conception d'ALDAC[19]	14
2.5	Flot de conception d'ALG[20]	14
2.6	Structure de l'outil de CAO ALADIN[24]	16
2.7	Structure de l'outil de CAO LAYGEN II[25]	16
2.8	Architecture d'Helix	17
2.9	Flot de développement des PCells dans Virtuoso	17
2.10	Générateur d'IP de CAIRO+	19
2.11	Architecture de CHAMS	20
2.12	Représentation chronologique des outils de CAO dédiés aux dessins des masques analogique entre 1985 et 2018	21
2.13	Description des outils de génération de dessins des masques pour circuits analogiques et mixtes depuis 1985	23
2.14	Description des outils de génération de dessins des masques pour circuits analogiques et mixtes depuis 1985(suite)	24
3.1	Ensemble des modules générables par Coriolis[1]	32
3.2	Contraintes de centrage géométrique (a) et de symétries (b)[3]	34
3.3	Placement sans (a) et avec (b) une prise en compte du sens du courant[4]	34
3.4	Profil thermique du circuit lorsque tous les modules sont répartis équitablement entre les quatre côtés (a) et deux côtés (b) du circuit[5]	35
3.5	Modules soumis à une contrainte de proximité (en rouge) pouvant être entourés d'un anneau de garde[3]	35
3.6	(a) Placement sans prise en compte de la régularité (b) Placement avec prise en compte de la régularité[8]	36
3.7	(a) Placement privilégiant des contraintes de symétries plutôt que des contraintes de régularité. (b) Placement tenant compte des contraintes de symétries et de régularité en même temps.[8]	36
3.8	Plan de masse irrégulier[9]	36

3.9	Deux exemples de placement d'un amplificateur opérationnel. (a) Les ports d'entrées sont placés au centre du groupe de modules symétriques (b) Les ports d'entrées sont placés en bordure du groupe de modules symétriques [10]	37
3.10	Exemple de placement et sa représentation en <i>slicing tree</i> où "H" définit une coupe horizontale et "V" une coupe verticale.	38
3.11	Exemple d'un placement représentable par un <i>slicing tree</i> (a) et d'un placement non représentable (b)	39
3.12	Description d'un échelon positif pour un module	39
3.13	(a) Échelons positifs résultant et $\Gamma_+ = ecadfb$. (b) Échelons négatifs résultant et $\Gamma_- = fcbead$. [18]	40
3.14	(a) Placement compact. (b) Représentation en <i>B*-tree</i> du placement compact (a).	40
3.15	(a) Une représentation en <i>O-tree</i> et (b) son placement correspondant. Pour ce placement de 8 modules, on a $T = 0010110010110011$ et $\pi = abcdefgh$	41
3.16	(a) Placement compact. (b) Représentation en <i>TCG</i> du placement compact (a).	42
3.17	Notre approche permet la génération rapide de plusieurs placements	46
3.18	Exemple d'écartement de placement autour du module D avec une représentation en <i>sequence pair</i>	47
3.19	Evolution des découpes (indiquées en rouge) d'un <i>slicing tree</i>	48
3.20	Exemple de création d'espaces dédiés aux fils de routage. (a) Le module concerné et le déplacement des nœuds aux alentours du nœud 6 (b) Résultat après écartement et en vert l'espace de routage créé autour du nœud 6	49
3.21	Transistor simple à 8 doigts et la représentation de ses attributs.	50
3.22	Représentation des attributs d'un nœud hiérarchique	52
3.23	Organisation en bande - Exemple de quatre modules (A, B, C et D) d'un nœud hiérarchique vertical	52
3.24	Représentation des attributs d'un espace de routage vertical et d'un espace horizontal	54
3.25	Hauteur et largeur d'un nœud hiérarchique vertical	55
3.26	Hauteur et largeur d'un nœud hiérarchique horizontal	55
3.27	Flot du placement analogique et mixte	58
3.28	Classes utilisées pour la construction du <i>slicing tree</i>	59
3.29	Parcours des facteurs de forme possibles	65
3.30	Classes utilisées pour évaluer et représenter les facteurs de forme	66
3.31	Facteur de forme d'un nœud hiérarchique et celui de ses nœuds fils	67
3.32	Interface utilisateur - Graphe de points des facteurs de forme	68
3.33	Interface utilisateur - Visualisateur	69
3.34	Interface utilisateur - Tableau des facteurs de forme	70
4.1	Exemple de graphe de routage global $G(S, A)$	80
4.2	(a) Arête avec une capacité de quatre pistes de routage (b) Arête avec une capacité de deux pistes de routage (c) Manque d'espace pour une piste de routage bien que les pavés soient adjacents.	81
4.3	Estimation de longueur d'un chemin dans un circuit numérique (a) et dans un circuit analogique (b)	81
4.4	Graphe de routage d'un circuit numérique (a) et d'un circuit analogique (b)	82

4.5	Représentation d'un <i>net</i> sur le graphe de routage d'un pavage numérique (a) et d'un pavage analogique (b)	83
4.6	Chemins connectant une source et une destination	84
4.7	Représentation de deux symétries : une paire de <i>nets</i> symétriques en rouge et un <i>net</i> dont les sommets sont symétriques en orange. Les sommets des pavés se trouvant sur l'axe de symétrie de ces deux <i>nets</i> sont représentés en vert	84
4.8	Occupation de pistes en fonction de la taille des fils. (a) est occupé par des fils avec un coût d'une 1 piste chacun (b) contient un fil avec une occupation de deux pistes de routage.	85
4.9	Restriction d'accès de sommets du graphe de routage	85
4.10	(a)Arbre minimal de Steiner, (b) arbres de Steiner minimal rectilinéaire et (c) un arbre couvrant	86
4.11	Exemple d'exploration exhaustive pour un <i>net</i> comprenant une source S , une destination D et un obstacle (X)	88
4.12	Estimation de la longueur de fils d'un <i>net</i> d'un circuit analogique en utilisant les distances centre à centre	91
4.13	Arêtes de type <i>horizontal</i> en bleu et <i>vertical</i> en rouge	91
4.14	<i>Slicing tree</i> d'un placement analogique (a) et la représentation de ses canaux de routage (b)	93
4.15	Déroulement de l'algorithme de Dijkstra avec deux sommets à connecter	95
4.16	Situations avec des sommets ponctuels et non ponctuels	96
4.17	Cas de composantes non ponctuelles	97
4.18	Étape 0 - Placement analogique (a) et son graphe de routage correspondant (b)	98
4.19	Étape 1 - Estimation de coût de chemin (a) et leur graphe de routage correspondant (b)	99
4.20	Étape 2 - Estimation de coût de chemin (a) et leur graphe de routage correspondant (b)	99
4.21	Étape 3 - Estimation de coût de chemin (a) et son graphe de routage correspondant (b)	100
4.22	Graphe de routage avec erreur d'estimation avec le chemin orange (a) et rouge (b)	100
4.23	Estimation de coût de chemin (a) et son graphe de routage correspondant (b)	101
4.24	Symétrie entre deux <i>nets</i>	102
4.25	Symétries entre sommets d'un <i>net</i> unique	102
4.26	Arbre d'interconnexions dans la fenêtre d'exploration	103
4.27	Flot du routage global mixte. Les étapes numériques et analogiques du flot sont indiquées en jaune et les étapes concernant uniquement les étapes analogiques sont indiquées en vert	104
4.28	Classes utilisées pour la construction du graphe de routage global	106
4.29	Exemple d'utilisation des méthodes $vcut(y)$ et $hcut(x)$	107
4.30	Exemple de création de pavage numérique	107
4.31	Création du pavage pour un nœud vertical d'un <i>slicing tree</i>	108
4.32	Exemple de graphe de routage mixte	108
4.33	Classes utilisées pour l'algorithme de Dijkstra	109
4.34	Étapes d'initialisation du graphe de routage	110
4.35	Étapes du déroulement de la recherche d'arbres d'interconnexions	113

4.36	Étapes de l'estimation du coût de l'arête en cours	114
4.37	Description de l'occupation d'un fil et de son coût en longueur dans un canal de routage horizontal	116
4.38	Gestion d'estimation de coût de longueur avec un <i>Vertex</i> destination et un <i>Vertex</i> source	117
4.39	Exemple de détection de présence de <i>VIAs</i>	117
4.40	Gestion d'estimation de coût de longueur entre un <i>Vertex</i> numérique et un <i>Vertex</i> analogique	118
4.41	Étapes de gestion de mémorisation de la référence	119
4.42	Description de l'occupation d'un fil avec la classe <i>Interval</i>	120
4.43	Exemple de chemin de <i>Vertex</i> entre un <i>Vertex</i> source (sommet inférieur droit en vert) et un <i>Vertex</i> destination (sommet inférieur gauche en rouge)	120
4.44	Mise à jour des intervalles suite à l'atteinte d'un nouveau <i>Vertex</i> destination	121
4.45	Matérialisation des fils de routage global pour un exemple numérique (a) et un exemple analogique (b) pour un <i>net</i> . Les connecteurs à joindre sont représentés en bleu.	123
4.46	Évolution de l'occupation de fils au sein d'un canal de routage horizontal après le traitement de quatre <i>nets</i> passant par cet canal de routage	124
4.47	Méthodes des classes du <i>slicing tree</i> dédiées à l'étape de routage global	124
4.48	Échanges d'informations entre la phase de placement et la phase de routage	127
5.1	Exemple de fil électrique, routage global	131
5.2	Exemple de fil électrique, routage détaillé	131
5.3	Exemple de contacts non-ponctuels et ponctuels	132
5.4	Exemple de décomposition en contacts ponctuels	133
5.5	Schéma du contact <i>terminal</i>	133
5.6	Schéma du coude (<i>Turn</i>)	134
5.7	Schéma de la branche horizontale (<i>HTee</i>)	134
5.8	Schéma de la branche verticale (<i>VTee</i>)	135
5.9	Contrainte des segments globaux	135
5.10	Exemple d'ensemble de segments alignés	136
5.11	Intervalle optimal, exemple avec perpendiculaires globaux seulement	137
5.12	Intervalle optimal, exemple avec perpendiculaires globaux et terminal	137
5.13	Exemple de création de <i>dogleg</i>	138
5.14	Exemple de changement de <i>layer</i>	139
5.15	Achèvement du routage d'un canal horizontal	140
5.16	Achèvement d'un <i>device</i>	141
5.17	Structure des pistes de routages	141
5.18	Coulissage d'un segment	142
5.19	Fonction de coût	145
5.20	<i>Rip-up</i> des perpendiculaires	146
5.21	Suite au <i>rip-up</i> des perpendiculaires	147
5.22	Résultat de la minimisation	148
5.23	Exemple de deux <i>nets</i> symétriques	149
5.24	Appariement des coûts	149
5.25	Exemple d'un <i>net</i> symétrique à lui-même	150
5.26	Prise en compte des segments larges	151
6.1	Flot de conception mixte à travers les outils logiciels du LIP6	154
6.2	Schéma électrique de l'amplificateur Miller à transconductance	155

6.3	Placement et <i>slicing tree</i> correspondant de l'amplificateur Miller	156
6.4	Pour le dimensionnement 1 - Graphe de points des placements possibles	158
6.5	Transistors de l'amplificateur Miller suite au routage global	159
6.6	Amplificateur Miller (sans la résistance et la capacité) placé et routé	159
6.7	Pour le dimensionnement 2 - Graphe de points des placements possibles	160
6.8	Routage global des transistors de l'amplificateur Miller, dimensionnement 2	161
6.9	Transistors de l'amplificateur Miller placés et routés, dimensionnement 2	161
6.10	Schéma électrique de l'amplificateur à source de courant ajustable	162
6.11	Schéma électrique de la polarisation	162
6.12	Décodeur numérique contrôlant la polarisation	163
6.13	Placement de l'amplificateur à source de courant ajustable	163
6.14	<i>Slicing tree</i> du placement de l'amplificateur à source de courant ajustable	164
6.15	Graphe de points des placements possibles	164
6.16	Amplificateur à source de courant ajustable placé et routé	165
6.17	Schéma électrique de la transconductance différentielle contrôlable	166
6.18	Circuit intermédiaire entre les sorties du décodeur et de l'entrée VC	166
6.19	Décodeur numérique contrôlant la tension d'entrée VC de la transconductance différentielle	167
6.20	Placement de la transconductance différentielle configurable	167
6.21	<i>Slicing tree</i> du placement de la transconductance différentielle configurable	168
6.22	Graphe de points des placements possibles	169
6.23	Transconductance différentielle configurable placée et routée	170

Liste des tableaux

2.1 De Cairo à Coriolis	22
3.1 Tableau récapitulatif de l'état de l'art des représentations de placement pour les circuits analogiques	43

Chapitre 1

Introduction

Sommaire

1.1 Contexte	2
1.2 Conception du dessin des masques mixtes actuelle	3
1.3 Enjeux et motivations	5
1.4 Contributions de cette thèse	6
1.5 Plan de la thèse	7
1.6 Références	8

1.1 Contexte

Au cours de ces dernières décennies, les circuits analogiques ont pris une place particulièrement importante dans les systèmes sur puce dont les applications sont diverses telle que la télécommunication, l'automobile, l'internet des objets ou encore l'ingénierie médicale. Bien que la plupart des fonctionnalités de ces circuits intégrés soient implémentées en circuit numérique, des fonctionnalités requièrent des circuits analogiques faisant le lien avec le monde extérieur. Une étude réalisée par *IC Insights*[1] prévoit une augmentation des ventes des circuits intégrés sur les cinq prochaines années à venir.

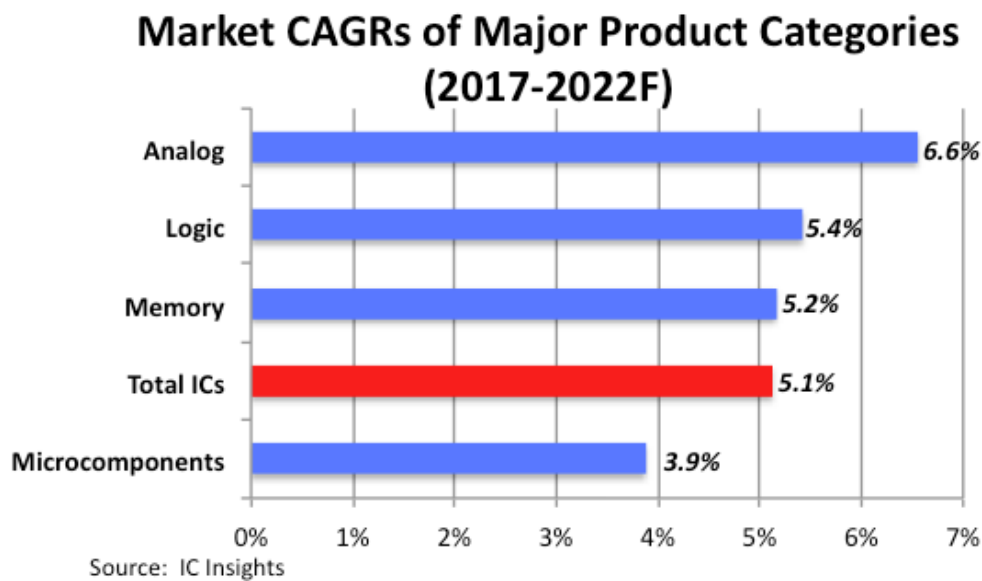


FIGURE 1.1 – Croissance des principales catégories de marché de circuits intégrés[1]

La figure 1.1 présente les estimations de l'étude sur la croissance des principales catégories de marché de circuits intégrés. On y observe en particulier que la croissance totale des ventes de tous les circuits intégrés augmentera de 5.1% et que celle des circuits analogiques est estimée comme étant la plus forte sur l'ensemble des marchés des circuits intégrés avec une croissance de 6.6%.

Malgré le besoin de circuits analogiques et de leur aptitude à être intégrés à des systèmes sur puce, les outils logiciels d'automatisation de conception de circuits analogiques restent bien moins avancés que ceux dédiés aux circuits numériques. Cela est dû à la complexité du problème de la conception de circuits analogiques :

- Il existe un grand nombre de classes de circuits spécifiques.
- Chaque classe de circuits nécessite sa propre approche de conception.
- Les circuits analogiques sont sensibles aux bruits et aux effets parasites liés aux procédés de fabrication.

La complexité de la conception de circuit analogique a pour conséquence que la conception reste "manuelle" afin de garder un maximum de contrôle sur l'ensemble des paramètres du circuit tout au long du flot de conception. En particulier, la conception du dessin des masques est une tâche de plus en plus fastidieuse avec l'augmentation du nombre

de règles de dessin des nouvelles technologies submicroniques. Il en résulte que ce processus est long et sujet à des erreurs. De manière générale, l'effort de conception de la partie analogique d'un circuit mixte est bien plus importante que celle de la partie numérique. Avec l'évolution des technologies de plus en plus fines couplée avec la constante augmentation de la demande, la conception de circuit analogique manuelle finira par nécessiter un effort de conception trop conséquent.

L'objectif de cette thèse consiste à proposer une méthodologie de conception de circuit mixte permettant d'accélérer la conception en automatisant certaines parties de la conception du dessin des masques mixte.

Avant de rentrer plus en détails dans notre approche, il est important de comprendre les limites du flot de conception actuel des circuits mixtes.

1.2 Conception du dessin des masques mixtes actuelle

Un circuit dit mixte est un circuit comportant des parties analogiques et des parties numériques. L'association de ces deux différents types de circuits est complexe car leur flot de conception est différent, en particulier en ce qui concerne la conception du dessin des masques. Le flot de conception du dessin des masques des circuits numériques et des circuits analogiques est présenté par la figure suivante :

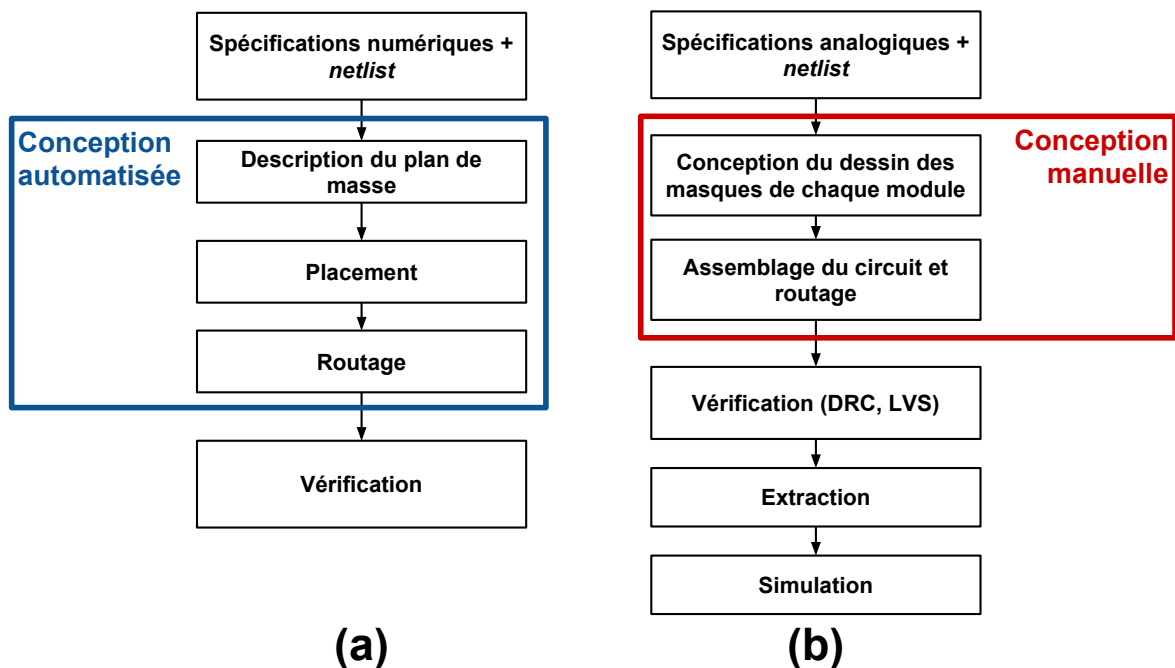


FIGURE 1.2 – (a) Flot de conception du dessin des masques numérique (b) Flot de conception du dessin des masques analogique

Sur la figure 1.2.(a), le flot de conception du dessin des masques des circuits numériques se déroule suivant ces étapes majeures :

- **Description du plan de masse** : Définition des dimensions du circuit comprenant des informations telles que la taille des fils d'alimentation.
- **Placement** : Placement de l'ensemble des cellules standard représentant le circuit.

- **Routage** : Routage du signal d'horloge et routage des *nets* du circuit en deux phases (routage global et routage détaillé).
- **Vérification** : Vérification des règles de dessin et des contraintes de timing du circuit.

Les étapes de placement et de routage des circuits numériques sont les principales étapes de la conception du dessin des masques et sont automatisées. Cela implique que la création et le positionnement des couches de métaux sont réalisés par le biais d'outils logiciels capables de placer et router ces couches de métaux à partir des règles de dessins et des contraintes de timing d'une technologie donnée.

Sur la figure 1.2.(b), le flot de conception du dessin des masques des circuits analogiques se déroule suivant ces étapes majeures :

- **Conception du dessin des masques de chaque module** : Création des fils de métaux réalisant la fonctionnalité de chaque module du circuit suivant les règles de dessins de la technologie donnée.
- **Assemblage du circuit et routage** : Placement du dessin des masques de modules et création des couches de métaux joignant les connecteurs des *nets*.
- **Vérification** :
 - **Vérification (DRC, LVS)** : Vérification des règles de dessin et de la correspondance avec le schéma électrique initialement désiré.
 - **Extraction** : Extraction des effets parasites liés aux procédés de fabrication.
 - **Simulation** : Simulation du circuit *post layout*.

L'ensemble des étapes de conception implique le dimensionnement et le positionnement de toutes les couches de métaux du circuit tout en prenant soin de respecter les règles de dessin et les contraintes d'électromigration. Ces deux tâches sont réalisées manuellement. Si durant l'une de ces étapes les spécifications ne sont pas respectées, il est nécessaire de revenir à l'étape précédente voire de devoir recourir à un redimensionnement du circuit entraînant une nouvelle redéfinition complète du dessin des masques. Ces itérations peuvent engendrer des efforts de conception importants et coûteux en temps.

Dans le cas de la conception de circuits mixtes, il n'existe pas de flot de conception unique utilisé pour la conception de tous les circuits mixtes mais il existe deux approches de conception du dessin des masques qui sont les suivantes :

- **Approche *digital-on-top*** : Cette approche consiste à employer le flot de conception numérique, en particulier les logiciels dédiés à la conception de circuit numérique, pour concevoir le circuit mixte. La partie analogique, préalablement placée et routée avec le flot de conception analogique, est introduite comme une boîte noire qui doit être placée au sein du plan de masse du circuit complet et être routée avec la partie numérique.
- **Approche *analog-on-top*** : Cette approche consiste à employer le flot de conception analogique pour concevoir le circuit mixte. Avec cette approche, la partie numérique, préalablement placée et routée avec le flot de conception numérique, est placée et routée avec la partie analogique manuellement. Il est également courant que les concepteurs souhaitent concevoir la partie numérique dans son intégralité en utilisant le flot de conception analogique uniquement.

Les concepteurs de circuits mixtes sont confrontés à une situation où ils doivent choisir entre un flot purement analogique et un flot purement numérique, chacun des deux ignorant l'autre. Les concepteurs ont complètement à leur charge de communiquer de façon ad-hoc des informations pertinentes entre les parties analogiques et numériques.

Suite à ces observations sur les limites de la conception de circuits analogiques et mixtes, on peut en déduire les problématiques suivantes :

- **Problématique 1** : La conception manuelle du dessin des masques analogiques devient de plus en plus complexe avec l'augmentation du nombre de règles de dessins pour les nouvelles technologies. Un circuit analogique doit également être à nouveau dimensionné en cas de spécifications non respectées mais également pour toutes nouvelles technologies utilisées rendant le dessin des masques d'un circuit analogique non réutilisable.
- **Problématique 2** : Chacune des approches de conception de circuits mixtes entraînent des pertes d'informations. Dans le cas de l'approche *digital-on-top*, il est difficile d'intégrer des contraintes analogiques à travers les outils logiciels dédiés au placement routage numérique. Dans l'approche *analog-on-top*, la conception de modules supplémentaires alourdit davantage la problématique précédente.

1.3 Enjeux et motivations

Le laboratoire d'informatique de Paris 6 (LIP6) s'intéresse au développement de méthodes de conception des composants numériques et analogiques pouvant être intégrés dans des systèmes sur puce depuis plus d'une vingtaine d'années. Le laboratoire a également lancé plusieurs *starts-ups* au cours de ces dernières années tels que FlexRas sur le partitionnement FPGA récemment acquis par Mentor Graphics, Intento Design sur la synthèse électrique et l'optimisation automatique des circuits analogiques, et Seamless Waves sur la radio cognitive. Ces faits montrent l'expérience en termes de conception de circuits numériques et analogiques au LIP6.

Le laboratoire est la source de plusieurs logiciels qui sont le fruit de travaux de thèse et de développement et sont dédiés à la réalisation de circuits numériques et de circuits analogiques :

- **Coriolis**[2] : pour le placement et le routage des circuits CMOS.
- **Alliance**[3] : un flot de CAO VLSI, incluant une bibliothèque de cellules standards.
- **OCEANE**[4] : pour le dimensionnement de fonctions analogiques intégrées sur silicium.
- **TAS**[5] : un outil d'analyse temporelle.

Initialement dédié aux circuits numériques, **Coriolis** comporte une extension de sa base de données dédiées à la conception de circuits analogiques appelé *CHAMS* (Cairo Hurricane AMS). Des travaux de thèse antérieurs ont enrichi cette base de données en y incorporant une bibliothèque de cellules analogiques permettant la génération du dessin des masques de blocs analogiques de base.

L'expérience de conception de circuits et de développement logiciels dédiés à la conception de circuits font partie des motivations pour proposer une solution aux problématiques de conception de circuits analogiques et mixtes.

1.4 Contributions de cette thèse

L'objectif de cette thèse consiste à répondre aux problématiques de la partie 1.2 en proposant une approche de conception de circuits mixtes. Pour cela, la contribution de cette thèse consiste à définir un environnement de conception assistée unifié pour les blocs numériques et analogiques au niveau dessin des masques. Il s'agira d'offrir au concepteur un contrôle fin des phases de placement et routage tout en garantissant une automatisation partielle des tâches et une communication fluides des informations d'une part entre le dimensionnement électrique et le dessin des masques, et d'autre part entre les blocs analogiques et les blocs numériques.

La méthodologie de conception de circuits mixtes que nous proposons repose sur l'uniformisation du flot de conception numérique et analogique. Nous jugeons nécessaire l'uniformisation de ces deux flots de conception dans le cadre de la conception de circuits mixtes. De par la nature du problème de la conception de circuit numérique, on souhaite préserver la distinction des étapes de placement, de routage global et de routage détaillé. C'est pourquoi notre approche de conception sera composée des étapes illustrées par la figure 1.3.

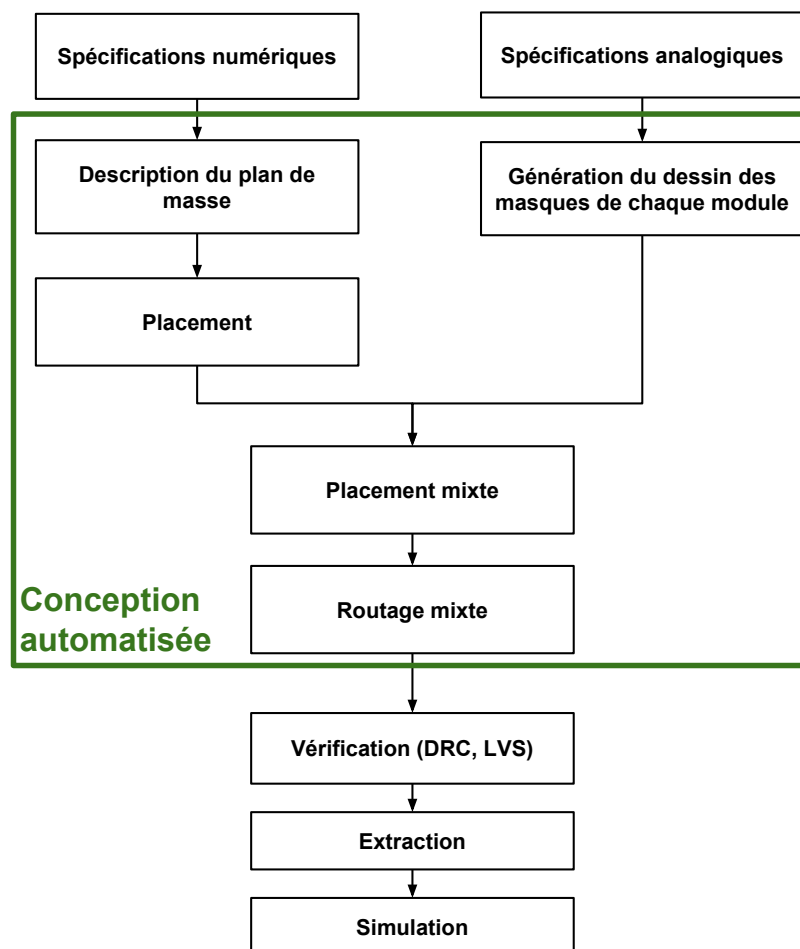


FIGURE 1.3 – Flot de conception mixte : unification du flot de conception numérique analogique

Unifier les flots de conception implique d'unifier les structures de données d'un point de vue logiciel. La qualité de l'architecture logicielle a un impact important sur l'exploita-

tion des ressources du circuit et l'efficacité des algorithmes exécutés. Notre architecture logicielle a été pensée avec l'optique de préserver la rapidité d'exécution des algorithmes de placement routage numérique qui nécessitent de manipuler un gros volume de données tout en étant capable d'appliquer des contraintes spécifiques aux circuits analogiques. Cela est réalisé à travers l'utilisation de classes communes qui sont enrichies pour les adapter aux problèmes du placement routage analogique.

Avec cette approche d'unification des deux flots de conception, nous souhaitons préserver au mieux l'efficacité des deux flots de conception. Le risque, en unifiant ces modes de conception, est de les détériorer en augmentant trop le volume de données à manipuler et le temps d'exécution. Les résultats présentés dans cette thèse montrent que ces risques sont limités.

L'objectif de cette thèse consiste à développer une méthodologie de placement routage mixte permettant de répondre aux problèmes liés au monde numérique et analogique afin d'unifier au mieux ces deux flots de conception. Les contributions de cette thèse sont les suivantes :

- **Placement analogique et mixte :**
 - Définition d'une méthodologie de placement analogique et mixte faisant intervenir l'expérience du concepteur et lui donnant du contrôle durant l'intégralité de la phase de placement.
 - Recherche de solution de placement en se basant sur la description d'un placement relatif à partir d'un *slicing tree* fourni par le concepteur comprenant en plus les contraintes de placement qu'il souhaite considérer.
 - Interface graphique permettant la visualisation des solutions de placement.
- **Routage global mixte :**
 - Transformation d'un placement obtenu en un graphe de ressources de routage en représentant le circuit sous la forme d'un pavage pouvant représenter un module analogique ou numérique ou un espace de routage.
 - Recherche de chemins d'interconnexions à travers les pavés pour l'ensemble des *nets* du circuit en utilisant un algorithme de Dijkstra adapté au problème du routage mixte. Estimation de longueurs de fils spécifiques aux *nets* analogiques à partir de la topologie du circuit en *slicing tree*.
 - Redimensionnement des canaux de routage à partir du *slicing tree*.
- **Routage détaillé :** Construction des fils de routage à partir du résultat du routage global. Création des éléments atomiques du routage détaillé manipulés pour la résolution des superpositions des fils.

1.5 Plan de la thèse

Ce manuscrit de thèse est organisé en 5 parties principales :

- **Chapitre 2 - Etat de l'art :** Ce chapitre présente l'état de l'art des outils logiciels d'automatisation de la conception analogique et mixte. L'ensemble des approches étudiées au cours de ces 20 dernières années y sont présentées et comparées afin de comprendre les approches précédemment étudiées. Le laboratoire d'informatique de Paris 6 est la source de plusieurs chaînes d'outils de CAO dédiées à la conception

de circuits intégrés. Un historique des travaux précédents y sera présenté afin de pouvoir situer le contexte dans lequel s'inscrit cette thèse .

- **Chapitre 3 - Placement** : Ce chapitre décrit le problème du placement mixte et les contraintes de placements les plus communément considérées. Un état de l'art spécifique aux approches de placement analogiques et mixtes y est décrit. On s'intéresse particulièrement au dessin des masques analogiques car elle est l'une des parties limitantes de la conception mixte. La description théorique et l'implémentation de notre méthodologie de placement mixte y est décrite ainsi que les raisons pour lesquelles nous avons choisi notre approche.
- **Chapitre 4 - Routage global** : Ce chapitre présente le problème du routage global et les contraintes à prendre en compte pour les *nets* numériques et les *nets* analogiques. La phase de routage global a été particulièrement étudiée à travers des travaux passés au LIP6, une étude de l'état de l'art du routage analogique y est présentée. Ce chapitre contient également la description théorique de notre approche de routage global mixte et de son implémentation logicielle.
- **Chapitre 5 - Routage détaillé** : Ce chapitre présente le routeur détaillé mixte et est le résultat de plus de 10 ans de travail dédié dans un premier temps aux circuits numériques. Ce routeur détaillé est particulièrement configurable et son fonctionnement a été enrichi pour prendre en compte les contraintes du routage analogique. Les travaux liés au routeur détaillé et réalisés dans le cadre de cette thèse y sont présentés ainsi qu'une description résumée du fonctionnement du routeur détaillé. Ce chapitre consiste à démontrer la capacité à utiliser le résultat du routage global pour obtenir un circuit complètement routé.
- **Chapitre 6 - Résultat** : Ce chapitre contient 3 circuits de tailles différentes sur lesquels notre approche de placement routage mixte a été utilisée afin de produire un dessin des masques placé et routé. Les étapes de placement et de routage mixtes y sont décrites en détails ainsi que les informations pertinentes telles que la surface occupée, le nombre de segments et contacts créés et la longueur total de fils de routage.

1.6 Références

- [1] IC Insight Research Bulletins. Analog ic market forecast with strongest annual growth through 2022, 2018. [ix, 2](#)
- [2] Christian Masson Gabriel Gouvine Sophie Belloeil Damien Dupuis Christophe Alexandre Hugo Clement Marek Sroka Jean-Paul Chaput, Rémy Escassut and Wu Yifei. Coriolis, 2018. [5](#)
- [3] Alain Greiner and François Pêcheux. Alliance : A complete set of CAD tools for teaching vlsi design. 1992. [5](#)
- [4] Jacky Porte. *OCEANE : Outils pour la Conception et l'Enseignement des circuits intégrés ANalogiques – reference manual*, January 2018. [5](#)
- [5] Anthony Lester Grégoire Avot Pirouz Bazargan-Sabet Amjad Hajjar, Karim Dioury and Marie-Minerve Louërat. Tas, 2018. [5](#)

Chapitre 2

État de l'art des outils de CAO pour circuits analogiques et mixtes

Sommaire

2.1 Introduction	10
2.2 Les outils de génération de dessins des masques	10
2.2.1 Les outils académiques	10
2.2.2 Les outils industriels	16
2.3 L'historique des outils de CAO au LIP6	18
2.3.1 Les travaux dédiés aux versions de CAIRO	18
2.3.2 Les travaux dédiés à CHAMS	19
2.4 Conclusion	21
2.5 Références	25

2.1 Introduction

Au cours des trois dernières décennies, de nombreuses approches ont été proposées dans le développement des outils de conception assisté par ordinateur (CAO) dédiés à l'automatisation de la génération du dessin des masques des circuits analogiques et mixtes. Ces approches d'automatisation proviennent majoritairement du milieu académique tandis que les outils industriels limitent l'automatisation à la génération d'un petit nombre de transistors paramétrables. Le laboratoire d'informatique de Paris 6 (LIP6) est la source d'outils de CAO dédiés à la conception de circuits intégrés. Des outils tels que Alliance ([1]), Coriolis ([2]), OCEANE ([3]) et Tas/Yagle ([4]) sont le résultat de plus d'une vingtaine d'années d'efforts et de travaux de recherches entrepris au sein du département System-On-Chip. Dans ce chapitre, nous présentons l'état de l'art des outils de CAO dédiés aux dessins des masques analogiques et mixtes ainsi que l'ensemble des travaux précédemment réalisés au LIP6 avant le début de cette thèse.

2.2 Les outils de génération de dessins des masques

Dans cette section, on présente les outils de CAO dédiés aux dessins masques provenant du milieu académique et du milieu industriel. On s'intéresse ici en particulier aux travaux présentant une génération complète d'un dessin des masques. Les travaux qui se focalisent uniquement sur une seule partie de la génération seront présentés dans les chapitres consacrés au placement et au routage.

2.2.1 Les outils académiques

Au cours des dernières décennies, les circuits intégrés analogiques ont gagné de plus en plus d'intérêt en majorité dû aux applications des systèmes sur puce (System-On-Chip) modernes qui sont des circuits comprenant des parties numériques et analogiques, autrement dit mixtes. Malgré cet intérêt porté aux circuits analogiques, les outils d'automatisation de la conception sont bien moins avancés pour l'analogique que pour le numérique. Cela s'explique par la difficulté à gérer toutes les contraintes de la génération du dessin des masques analogique. Par conséquent, l'état de l'art s'oriente davantage sur des recherches d'automatisation pour circuits analogiques que pour circuits mixtes.

Les études de l'automatisation de la génération du dessin des masques des circuits analogiques et mixtes se focalisent sur deux étapes communes qui sont la phase de placement et la phase de routage. C'est en utilisant des modules analogiques pouvant être un transistor simple ou un petit bloc analogique réalisant une fonction simple (miroir de courant, paires différentielles, etc ...) que ces outils réalisent la phase de placement et de routage. Les outils modernes traitant ces sujets peuvent être classés selon l'approche suivie :

- **Génération du dessin des masques analogiques et mixtes à partir de contraintes :** À partir du schéma électrique du circuit et des contraintes de placement et de routage spécifiées par le concepteur, ces outils génèrent les modules analogiques, les placent de manière optimisée et déterminent le routage en minimisant la longueur des fils. De nombreux articles de l'art ont contribué à cette approche de génération de dessin des masques et portent sur une de ces trois étapes en incorporant la prise en compte de contraintes (symétries, appariement de modules, routage symétrique, etc ...).

- **Migration technologique du dessin des masques pour de nouvelles spécifications :** La réalisation du dessin des masques d'un circuit analogique requiert l'expérience et les connaissances adéquates. Le niveau d'expertise et les connaissances d'un concepteur influent de manière conséquente sur la qualité du dessin des masques. Par conséquent, l'exportation d'un circuit analogique d'une technologie à une autre est l'objectif d'une classe d'outils de CAO. Ces outils cherchent à capitaliser sur l'expertise des concepteurs afin de l'adapter pour une nouvelle technologie ou selon de nouvelles spécifications.

Que ces outils soient destinés à la génération du dessin des masques à partir de contraintes ou à la migration technologique, ils abordent le problème de la génération du dessin des masques en utilisant différentes approches. On classe les approches des outils de l'état de l'art en 3 catégories se distinguant par les méthodes de résolution utilisées pour aborder le problème du placement routage analogique et mixte :

- **Les outils procéduraux :** On considère un outil procédural comme étant un outil qui permet au concepteur de décrire l'ensemble des étapes de placement et de routage. À l'aide d'un langage dédié, le circuit est entièrement décrit par un script indiquant l'emplacement de chacun des modules analogiques et des fils permettant la connectivité électrique. De cette manière, les outils procéduraux ont l'avantage de donner le contrôle au concepteur de la génération du circuit. L'exécution de ces outils est rapide car elle ne nécessite pas de temps de recherche de solutions et permettent une reproductibilité du circuit. Les scripts de ces outils procéduraux ont le désavantage d'être longs à décrire et ne sont valables que pour une technologie donnée. De plus, ils ne proposent pas d'aide au concepteur à la recherche de solutions de placement ou de routage.
- **Les outils utilisant des méthodes d'optimisation :** Cette dernière catégorie rassemble les outils fondés sur des méthodes métaheuristiques pour trouver une solution de placement routage. Le problème du placement et du routage nécessite de parcourir un très grand espace de solution et ces méthodes permettent d'obtenir une bonne solution rapidement. L'avantage de cette approche repose sur une mise en place rapide en laissant l'outil la charge de réaliser le choix du placement routage dans un espace de solutions souvent très grand. En revanche, l'exploration de l'espace de solutions peut s'avérer longue. L'ajustement des fonctions de coût peut s'avérer difficile pour le concepteur, la reproductibilité n'est pas toujours possible.
- **Les outils se basant sur un *template* :** Ces outils ont pour but de permettre au concepteur de partir d'une base pour réaliser un circuit. Un *template* est un patron incorporant les informations permettant de représenter une topologie de placement ou de routage basées sur l'expérience d'un concepteur. Ce *template* est par la suite optimisé ou réajusté afin de correspondre au besoin du circuit qu'on souhaite réaliser en particulier pour la réutilisation du circuit pour une nouvelle technologie. Dans la mesure où un *template* n'est pas capable de représenter le circuit désiré, le concepteur est obligé de créer un nouveau *template* ce qui peut être long et complexe. Ces outils incluent des méthodes d'optimisation pouvant optimiser la phase de placement ou de routage.

Le tableau de la figure 2.1 résume l'ensemble des avantages et des inconvénients que présente chacune de ces approches. Chaque approche possède son intérêt et il est difficile de déterminer laquelle de ces approches domine les autres en termes de qualité du placement routage de manière générale pour un temps raisonnable. Il est possible en revanche d'établir des critères de qualité permettant de juger individuellement la capacité

	Avantages et Inconvénients des approches de génération de dessins des masques analogiques		
	Procédural	Optimisation	Template
Avantages	<ul style="list-style-type: none"> • Paramétrage donnant un control total au concepteur • Temps d'exécution court • Reproductibilité du placement-routage 	<ul style="list-style-type: none"> • Haut niveau d'abstraction • Utilise des méthodes d'optimisation • Temps de configuration rapide pour un nouveau circuit 	<ul style="list-style-type: none"> • Plus haut niveau d'abstraction qu'en procédural • Incorpore l'expérience d'un concepteur • Ré-utilisabilité pour une nouvelle technologie
Inconvénients	<ul style="list-style-type: none"> • Configuration longue du script pour un circuit entier • Non ré-utilisable pour une autre technologie donnée • Pas d'utilisation de méthodes d'optimisation 	<ul style="list-style-type: none"> • Longue durée d'exécution • Peu de contrôle par le concepteur sur le résultat final • Petites modifications difficiles • Reproductibilité pas toujours possible 	<ul style="list-style-type: none"> • Requièr une 1ere description complète du template • Difficile de convertir en template toutes les topologies • Ne concerne que le placement

FIGURE 2.1 – Comparaison des approches des outils de génération de dessins des masques pour circuits analogiques et mixtes

d'un outil à pouvoir générer le dessin des masques voulu par le concepteur.

Un outil de génération de dessins des masques analogiques et mixtes doit être en mesure à répondre au besoin du concepteur, c'est-à-dire être capable de représenter des topologies variées. La génération doit pouvoir se faire dans un temps raisonnable tant pour la phase de paramétrage que le temps d'exécution. On définit la qualité d'un outil de génération de dessins des masques à partir des critères suivants :

- **Types de contraintes gérées** : La qualité d'un dessin des masques, en particulier dans le cas des circuits analogiques et mixtes, est influencée par la prise en compte de contraintes de conception. Ces contraintes sont diverses et ont toutes pour but de limiter les effets parasites dus au dessin des masques et de préserver les performances du circuit. Ces contraintes concernent la phase de placement et de routage, les plus communes sont les contraintes d'appariement et de symétries. Les détails concernant l'ensemble des contraintes existantes sont respectivement présentés dans les chapitres de la thèse dédiés au placement et au routage.
- **Catégorie de circuits réalisables** : En particulier pour les circuits analogiques, certains outils de l'état de l'art se concentrent sur la conception d'une catégorie de circuit. Le dessin des masques d'un circuit radiofréquence ou d'un convertisseur analogique numérique présentent des différences en termes de contrainte de placement et de routage. La polyvalence et la spécialisation d'un outil présentent chacune des avantages et des inconvénients. La polyvalence permet de réaliser une plus grande diversité de circuit mais au risque de proposer une solution moins bonne. En opposition, la spécialisation permet de prendre en compte davantage de facteurs pour obtenir une meilleure solution mais risque de devenir obsolète pour les technologies futures en cas de spécialisation trop avancée.

- **Temps de conception** : Le niveau d'automatisation est un facteur important pour les outils de génération de dessin des masques de circuits analogiques et mixtes. Ce critère correspond au temps nécessaire pour que le concepteur interprète le circuit à réaliser au sein de l'outil. Ce temps de préparation combiné au temps d'exécution nécessite d'être significativement inférieur à une génération du dessin des masques manuelle. Malgré le grand nombre de contraintes imposées aux circuits analogiques, l'espace de solution pour un circuit donné reste toujours particulièrement grand et il est inconcevable que le temps d'une unique exécution dure plusieurs semaines.
- **Fonctionnalités de conception** : On définit une fonctionnalité comme toute aide supplémentaire apportée au concepteur. Celle-ci peut prendre plusieurs formes. Des aspects logiciels permettant une interactivité avec le circuit, la capacité à mémoriser la conception d'un script, une représentation de l'ensemble des données pertinentes du circuit ou bien l'interfaçage avec des outils logiciels extérieurs sont des fonctionnalités utiles dans l'accompagnement du concepteur dans l'étape de réalisation du dessin des masques.

L'étude du problème de l'automatisation de la génération du dessin des masques des circuits analogiques et mixtes débute avec la publication des premiers articles dans les années 1980. C'est en gardant en perspective l'ensemble des aspects mentionnés ci-dessus que nous présentons une vue d'ensemble de l'état de l'art de ces outils de CAO.

Les premières approches ayant pour but d'automatiser la génération du dessin des masques mixtes analogiques consistaient à générer de manière procédurale des modules [5]. Ces générateurs procéduraux définissent la représentation géométrique de la totalité du circuit. *ALSYN*[6] est capable de synthétiser le dessin des masques à partir d'une description niveau *netlist* pour une variété de circuits analogiques. L'outil avait pour but d'être flexible et de laisser beaucoup de contrôle au concepteur à travers un environnement de conception interactif. La figure 2.2 illustre le flot de conception d'*ALSYN*.

L'outil de Xu Jingnan et al.[8] utilise une librairie de modules paramétrables analogiques (*PCells*) décrits à partir du langage *SKILL*[9]. L'outil est développé à travers le Cadence Design Framework[10] et a pour but de générer des circuits analogiques et mixtes indépendamment des la technologie. Malgré la rapidité des outils procéduraux, ces outils manquent de flexibilité et le coût en temps de conception peut s'avérer long pour

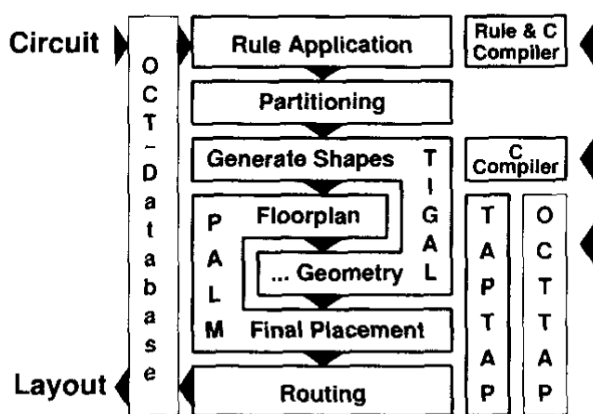


FIGURE 2.2 – Structure et flot de conception d'*ALSYN*[6]

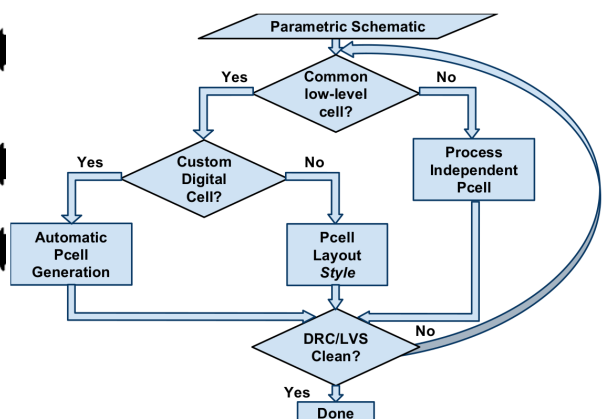


FIGURE 2.3 – Flot du générateur de dessins des masques de *BAG*[7]

la conception d'un nouveau circuit et la migration technologie de certains circuits peut entraîner des modifications. D'autres approches plus récentes cherchent également toujours à exploiter les *PCells*[11].

IIP Framework[12] est un environnement dédié au développement de générateurs de circuits analogiques indépendants de toute technologie et d'environnement de conception. L'outil consiste à paramétrer des descriptions de chaque vue d'un module analogique, c'est-à-dire la vue schéma électrique, *symbolique* et *en dessin des masques*. En particulier, leur objectif revient à permettre à la réutilisation de circuits analogiques.

BAG[7] est un environnement dédié à la génération de circuits analogiques et mixtes soumis à des contraintes de spécifications et indépendamment d'une technologie. À partir d'un ensemble de classes, la codification des procédures de dimensionnement et la paramétrisation du dessin des masques sont décrites par le concepteur. La phase de placement routage est réalisée à partir de ces classes contenant différentes architectures et méthodes de routage. La figure 2.3 illustre le flot de conception de *BAG*.

ILAC[13] fait partie des premières études utilisant l'approche basée sur des techniques d'optimisation. *ILAC* est un outil issu d'un ensemble d'outils d'*IDAC*[14] et utilise l'algorithme du recuit simulé[15] en tenant de contraintes d'appariement, de symétries et de proximité.

KOEN/ANAGRAM II[16] est un outil rassemblant un outil de placement (*KOEN*) et un outil de routage (*ANAGRAM II*). Le placement est obtenu à partir de modules provenant d'une librairie de générateurs de modules et le routage est réalisé en routant au-dessus des modules et respectant des contraintes de symétries.

LAYLA[17] tient compte des contraintes de symétrie, des dégradations de performance dues aux effets parasites et de l'appariement des modules pour optimiser le placement du circuit. Tout comme pour *KOEN/ANAGRAM II* et *ILAC*, la technique d'optimisation utilisée est l'algorithme de recuit simulé. *LAYLA* a également été utilisé par l'outil *CYCLONE*[18] ayant pour but de dimensionner et générer des oscillateurs LC.

ALDAC[19] applique une méthodologie suivant trois points clés (voir figure 2.4). La

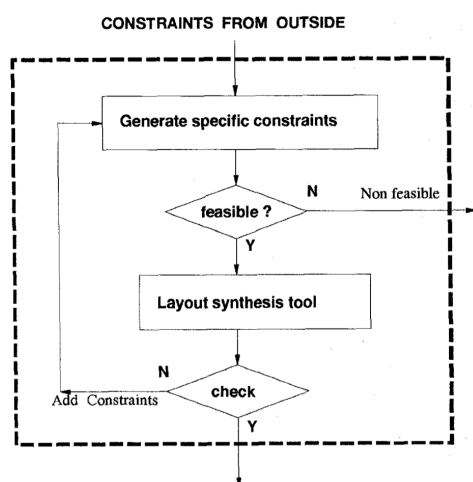


FIGURE 2.4 – Flot de conception d'*ALDAC*[19]

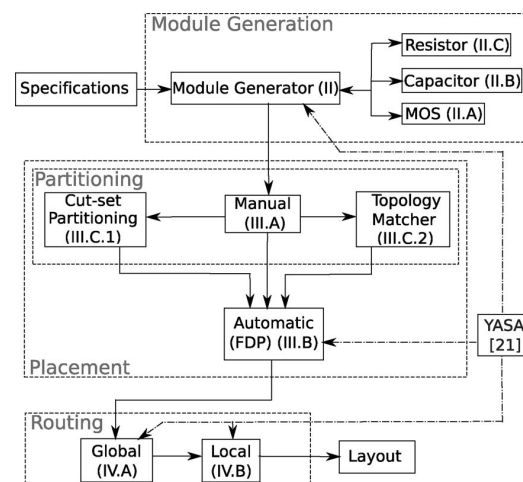


FIGURE 2.5 – Flot de conception d'*ALG*[20]

transformation de contraintes de spécifications en un jeu de contraintes garantissant la faisabilité de la génération. Les contraintes d'un haut niveau d'abstraction sont traduites en un jeu de contraintes sur les paramètres bas-niveau. Toute étape irréalisable est détectée au plus tôt du flot de conception.

L'outil *ALG*[20] se divise trois parties : un générateur de modules, un placeur et un routeur (voir figure 2.5). La première étape consiste à générer l'ensemble des modules à partir des informations des données d'entrée qui sont la *netlist* du circuit et les spécifications. Ces modules sont ensuite placés avec plus ou moins d'automatisation selon le choix du concepteur. Le routeur est décomposé en deux étapes, une étape de routage global et une étape de routage local/détaillé.

Le flot de conception de l'outil de Husni Habal et al.[21] est dirigé par le schéma électrique et une liste de paramètres du circuit. Chaque dessin des masques pour chacun des modules est considéré et la configuration de modules proposant les meilleures spécifications géométriques est conservé. Le placement et le routage sont réalisés tout en préservant la faisabilité du circuit ainsi que la minimisation des effets parasites dus au routage. La solution obtenue est réutilisée pour de réajustements du dimensionnement.

DeMixGen[22] se concentre sur la conception de circuits mixtes, en particulier des modulateurs sigma-delta. Cet outil a pour but de résoudre les problèmes de bruit entre les signaux numériques et analogiques. La génération des circuits mixtes est réalisée en portant une attention particulière à la séparation de ces deux types de signaux. À la différence des autres outils d'optimisation, le concepteur fournit la *netlist* du circuit qui est analysée pour réaliser le routage adéquat.

En parallèle aux approches basées sur l'optimisation, les approches basées sur des *templates* ont connu autant de succès que les précédentes approches pour leur capacité à intégrer l'expérience des concepteurs. Les premiers outils basés sur des *templates* avaient pour objectif de résoudre les problèmes de migrations technologiques.

IPRAIL[23] est un outil capable de redéfinir le dessin des masques de circuits analogiques existant pour une autre technologie ou pour de nouvelles spécifications. L'approche employée consiste à créer automatiquement un *template* à partir du circuit existant et imposer les nouvelles règles et spécifications aux modules redimensionnés.

ALADIN[24] est composé de trois parties qui sont : un environnement de générateur de module, un assistant de conception et une interface de technologie (voir figure 2.6). Le générateur de modules permet au concepteur de décrire, indépendamment de la technologie et de l'application visée, la paramétrisation des modules. L'assistant de conception est intégré au Cadence FrameWork[10] et fournit une interface graphique à l'utilisateur pour optimiser la phase de conception. L'interface de technologie permet la description des règles technologiques ainsi que des contraintes spécifiques au circuit.

Gönenç Berkol et al.[26] proposent un flot de conception de circuits analogiques comprenant un outil de génération du dessin des masques. Le dessin des masques est instancié à partir d'un *template* et l'occupation de l'espace est minimisée en changeant des paramètres géométriques tels que le nombre de doigts d'un transistor. Le *template* est décrit en langage LDS[27] et contient le code du placement et du routage.

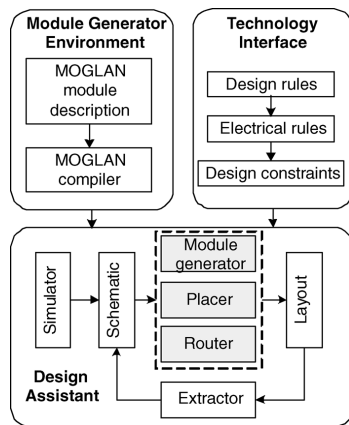


FIGURE 2.6 – Structure de l'outil de CAO ALADIN[24]

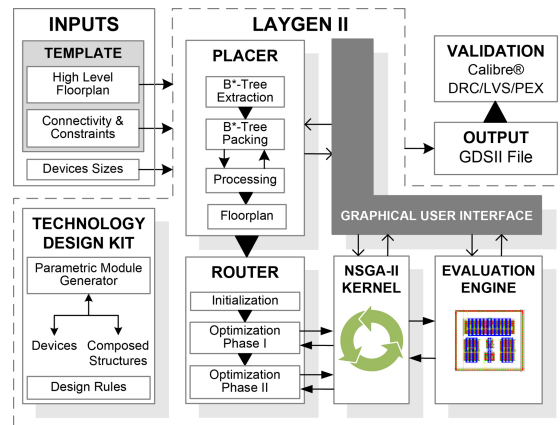


FIGURE 2.7 – Structure de l'outil de CAO LAYGEN II[25]

LAYGEN[28], puis amélioré en *LAYGEN II*[25] fait partie de l'ensemble d'outils *AIDA*[29] destinés à plusieurs niveaux de conception analogique. *LAYGEN II*, dédié à la génération de dessins des masques analogiques, utilise l'approche à base de *template* afin de représenter des topologies de placement et de routage provenant de l'expérience de concepteurs. La topologie choisie est ensuite optimisée et adaptée à la conception du circuit. Le routage prend en compte des contraintes de symétries et de sensibilité. La figure 2.7 illustre la structure de *LAYGEN II*.

2.2.2 Les outils industriels

Le développement d'outils de CAO dédiés à la génération du dessin des masques pour circuits analogiques et mixtes a également été étudié dans le monde industriel. La majorité de ces outils sont issus des 3 plus importantes entreprises du marché de l'automatisation de la conception électronique, c'est-à-dire *Synopsys*[30], *Cadence Design Systems*[31] et *Siemens-Mentor*[32]. Ces entreprises ont acquis au cours de ces dernières années des entreprises du domaine telles que *Ciranova*[33], *Magma Design Automation*[34], *Neoliner*[35] ou encore *Taner EDA*[36].

Ciranova Helix, à présent acquis par *Synopsys* et incorporé dans l'outil *Custom Compiler*[37], est un outil de placement supporté par une interface graphique. À partir d'une *netlist*, le concepteur décrit de manière hiérarchique le placement en précisant des contraintes géométriques et d'appariement. L'outil évalue ensuite l'ensemble des combinaisons possibles et propose de multiples solutions de placement DRC correctes résultant des contraintes introduites. Un fichier de sortie décrit en *OpenAccess* permet l'interfaçage avec d'autres outils de CAO commerciaux communément utilisés. Le temps de réalisation du placement final en technologie CMOS 65nm varie entre une demi-journée pour un comparateur comprenant 40 modules et 4 jours et pour une PLL composé de 1975 modules. La figure 2.8 présente l'architecture de l'outil.

Synopsys développe également des cellules paramétrables appelées *Python Parameterized Cells (PyCells)* pouvant être utilisées à travers une interface graphique. *PyCell Studio* regroupe un ensemble d'outils construits permettant le développement des *PyCells* : un

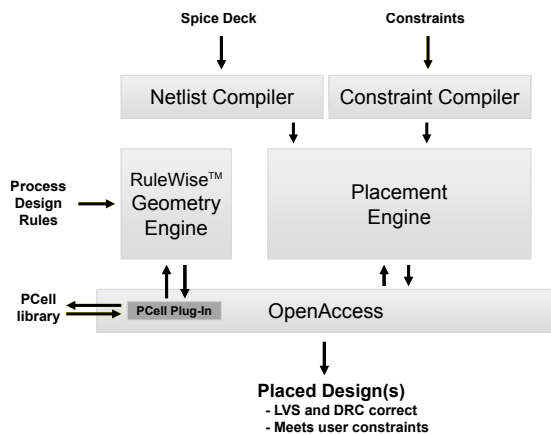


FIGURE 2.8 – Architecture d’Helix

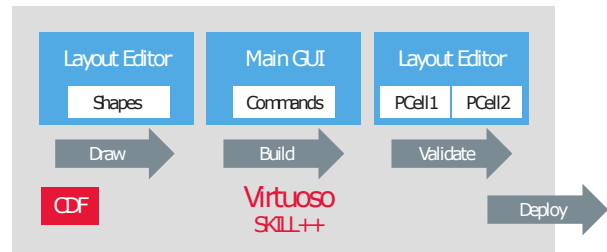


FIGURE 2.9 – Flot de développement des PCCells dans Virtuoso

visualiseur de dessin des masques *OpenAccess*, un environnement *Python shell* permettant l’exécution de ligne de commande *Python*, un environnement de conception intégré pour le débogage et un plug-in *OpenAccess* permettant la génération de formes géométriques. La figure 2.9 décrit le flot de développement des PCCells.

IRoute et *ARoute*[38] de *Mentor Graphics*[32] sont des outils de routage pour circuits analogiques et mixtes permettant un accompagnement interactif de la génération du routage. Le concepteur est amené à choisir l’ordre des *nets* à router définissant ainsi différents niveaux de priorité pour ces derniers. Les *nets* possédant une plus haute priorité seront routés en premier et plus directement. Les contraintes de largeur de fils ainsi que d’espace entre fils de nets différents sont fixées par le concepteur. *ARoute* présente une fonction de *rip-up and reroute* pouvant être activée ou non selon le choix du concepteur tandis que *IRoute* permet un routage à la main interactif du concepteur. L’outil présente également des marqueurs et des directives accompagnant le concepteur de manière interactive dans la phase de routage.

Tanner EDA Place and Route[39], à présent acquis par *Mentor Graphics* et incorporé dans *Taner L-Edit IC*[40], est un outil de placement routage pour circuit numérique au sein de circuits mixtes conçu avec une approche *Analog on Top*. Une interface graphique permet de placer et router rapidement des modules de contrôle d’un convertisseur analogique numérique en important les fichiers technologiques. Le concepteur définit les contraintes du plan de masse comprenant la taille de la puce et routage d’alimentation. Pour les modules analogiques, l’outil analyse la *netlist* et reconnaît les miroirs de courant et les paires différentielles. Le concepteur possède le contrôle sur l’ensemble de la génération des modules analogiques, placement et routage avec un paramétrage particulier concernant les miroirs de courant et les paires différentielles afin de limiter les effets parasites et s’assurer du bon appariement.

Les *Parameterized Cells (PCells)*[41] de *Cadence Design Systems* sont des cellules configurables pouvant être utilisées pour la génération de modules analogiques. L’utilisation des *PCells* passe à travers plusieurs étapes de conception : la définition des formes du devices, la paramétrisation des formes et des dépendances entre les formes et les couches de métaux, le débogage et la validation des *PCells*. Le rendu des *PCells* est visualisable à travers une interface graphique montrant les changements apportés en temps réel. La paramétrisation des *PCells* peut également être réalisée à travers des lignes de commandes

en *SKILL* ce qui a également été exploité par certains outils académiques [8][11].

Neoliner[35] acquis par *Cadence Design Systems* comporte un ensemble d'outils dédiés à l'automatisation de la génération du dessin des masques de circuits analogiques. Un routeur analogique interactif permet de router automatique les fils dont le résultat peut être corrigé par le concepteur s'il le considère invalide. Il est également possible de router symétriquement et de préciser le niveau de sensibilité d'un *net* afin que le routeur puisse trouver le chemin le plus court pour les nets les plus sensibles. Il existe un générateur de modules compacts pouvant être utilisé pour des technologies avancées.

2.3 L'historique des outils de CAO au LIP6

2.3.1 Les travaux dédiés aux versions de CAIRO

CAIRO, *CAIRO2* et *CAIRO+* (*Creating Analog IPs- Reusable and Optimized*) sont le résultat de nombreux efforts réalisés entre 1996 et 2008 par des travaux de thèses de contributeurs :

- Mohamed Dessouky[42], "*Conception en vue de la réutilisation de Circuits Analogiques. Application : Modulateur Delta-Sigma à très Faible Tension*", Doctorat, Université Pierre et Marie-Curie, France, Janvier 2001
- Pierre Nguyen Tuong[43], "*Définition et implantation d'un langage de conception de composants analogiques réutilisables*", Doctorat, Université Pierre et Marie-Curie, France, Juin 2006
- Laurent De Lamarre[44], "*CAIRO+ : Intégration du modèle Bsim3v3, stage de DEA*", Doctorat, Université Pierre et Marie-Curie, France, 2006
- Vincent Bourget[45], "*Conception d'une Bibliothèque de Composants Analogiques pour la Synthèse Orientée Layout*", Université Pierre et Marie-Curie, France, Novembre 2007
- Ramy Iskander[46], "*Synthèse de composants analogiques intégrés VLSI réutilisables*", Université Pierre et Marie-Curie, France, Février 2008

CAIRO+ est un environnement de conception basé sur un langage de description permettant la migration technologique d'une propriété intellectuelle (IP) d'une technologie vers une autre.

L'architecture du générateur d'IP analogique est présentée par la figure 2.10 et repose sur 4 parties :

- **Create** : Spécification d'une *netlist* pour un schéma électrique non dimensionné, d'un dessin des masques pour un placement relatif et d'une interface fonctionnelle de paramètres d'entrées et de sorties.
- **Design Space Exploration** : Description de procédures de dimensionnement, exploration de l'espace de solutions et évaluation des performances.
- **Shape and Place** : Évaluation des configurations possibles du dessin des masques et choix du dessin des masques respectant au mieux les contraintes géométriques de largeur et de hauteur.
- **Route** : Routage procédural réalisé pour le dessin des masques généré et annotation des parasites sur la *netlist*.

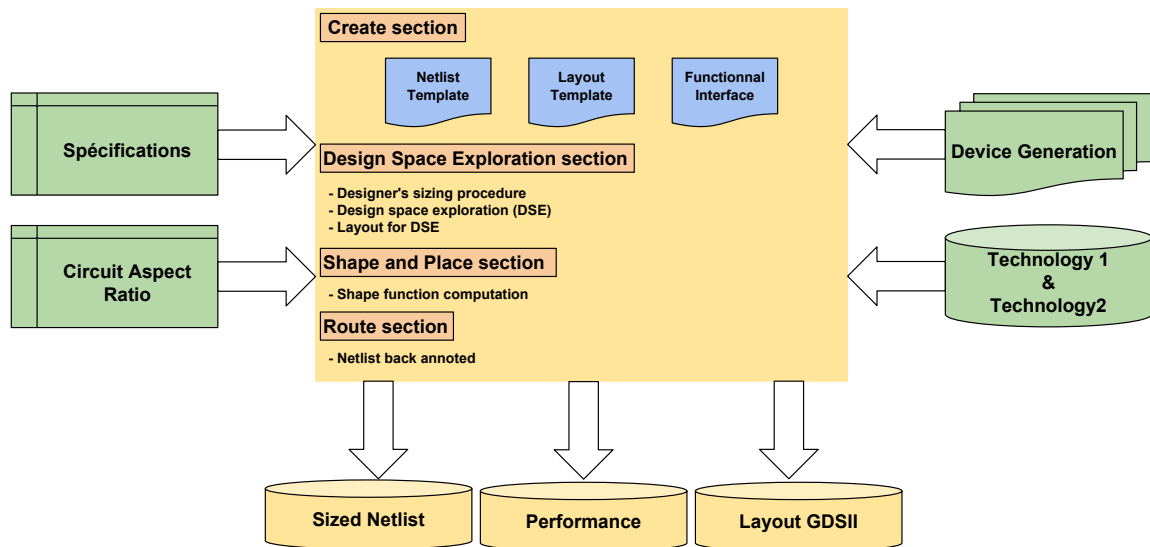


FIGURE 2.10 – Générateur d'IP de CAIRO+

Les explications de la génération du dessin des masques sont détaillées dans la thèse de Vincent Bourget[45]. Son travail est lié au développement de modules analogiques de base pour lesquels le dessin des masques est synthétisé. Néanmoins, l'approche utilisée comprenait quelques limitations :

- Le dessin des masques des modules était basé sur une grille réelle mais placé par aboutement sur la grille symbolique. La vue symbolique implique un dessin des masques dessiné avec des rectangles virtuels déterminant la boîte englobante des connecteurs tandis que pour la vue réelle, le dessin des masques est dessiné de façon à être prêt pour la fabrication.
- Les calculs des paramètres parasites du dessin des masques ne prenaient pas en compte les effets nanométriques récents tels que les effets de proximité et de stress.
- Le dessin des masques du circuit comprenait de long canaux de routage tout au long des composants d'un module augmentant ainsi les effets parasites et limitant les performances.
- Aucun algorithme d'optimisation n'était utilisé pour optimiser le placement du circuit.

2.3.2 Les travaux dédiés à CHAMS

Le projet CHAMS (*Cairo Hurricane AMS*) a pour but de combiner les bases de données de CAIRO et Hurricane, qui sont dédiées respectivement aux dimensionnement des circuits analogiques et à la conception de circuits intégrés VLSI. CHAMS est également le résultat de travaux de thèses :

- Stéphanie YOUSSEF, "*Conception d'une bibliothèque de cellules analogiques*", Université Pierre et Marie-Curie, France, Décembre 2012
- Farakh JAVID, "*Synthèse structurée des circuits analogiques intégrés capitalisant la connaissance du concepteur : vers une perspective industrielle*", Université Pierre et Marie-Curie, France, Avril 2013

CHAMS propose une définition d'un IP analogique sous la forme d'une structure paramétrable relative à une fonction particulière (i.e. amplification, filtrage ou conversion)

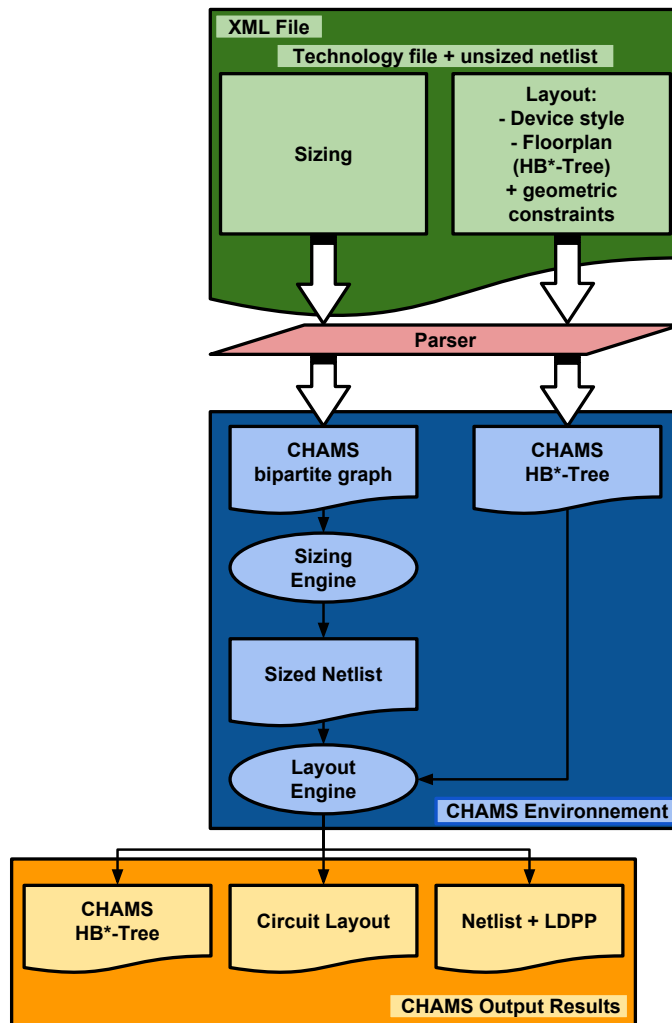


FIGURE 2.11 – Architecture de CHAMS

associée à une méthode de calcul des paramètres et à une génération automatique du dessin des masques de cette structure. Cette approche permet ainsi d'utiliser la même structure dans diverses applications et de la réaliser avec différents procédés technologiques cibles, pourvu qu'ils soient CMOS.

CHAMS propose de décrire la structure en terme de hiérarchie de sous-circuits. Les feuilles de la hiérarchie sont des cellules paramétrées. Il s'agit de composants élémentaires passifs tels que résistances et capacités mais aussi de composants élémentaires actifs tels que transistors NMOS et PMOS. En plus de ces composants élémentaires simples, les feuilles de la hiérarchie sont constituées de petits ensembles de composants fortement liés par des contraintes électriques et donc soumises à de fortes contraintes lors du dessin des masques. Il s'agit de la paire différentielle et des miroirs de courant pour les composants actifs et des matrices de capacités pour les composants passifs. Pour accélérer le processus de synthèse, *CHAMS* introduit une forte interaction entre le comportement électrique des composants de base et leur génération du dessin des masques.

2.4 Conclusion

De nombreux outils de CAO dédiés à la génération des dessins masques analogiques et mixtes ont été développés au cours de ces dernières années. Ce sujet, actuellement populaire, devient de plus en plus critique avec l'augmentation de la complexité tant des circuits que des contraintes de conception. L'état de l'art présente des outils ayant différents objectifs. Ces outils sont capables de générer un dessin des masques à partir de contraintes de placement et de routage pour concevoir de nouveaux circuits. D'autres outils cherchent à reproduire des circuits déjà réalisés pour une technologie donnée vers une technologie ou spécification différente.

On observe différentes stratégies pour la génération du dessin des masques que ce soit pour la génération de nouveaux circuits ou l'adaptation d'anciens circuits. Les approches les plus communes des outils académiques et industriels sont les approches procédurales, les approches se basant sur des méthodes d'optimisation et les approches basées sur un *template*. Chaque approche présente ses propres atouts et inconvénients et elles sont globalement équitablement utilisées.

Cette thèse a pour but de proposer une contribution supplémentaire au placement-routage mixtes. À notre connaissance, très peu d'études ont été réalisées dans le cadre du placement routage mixte qui de manière générale reste séparé pour la partie numérique et la partie analogique. Notre approche vise à trouver l'équilibre entre donner du contrôle au concepteur et automatiser les étapes fastidieuses et sources d'erreurs humaines. Les figures 2.12, 2.13 et 2.14 présentent chronologiquement l'ensemble des outils de l'état de l'art et brièvement les différentes caractéristiques de ces outils.

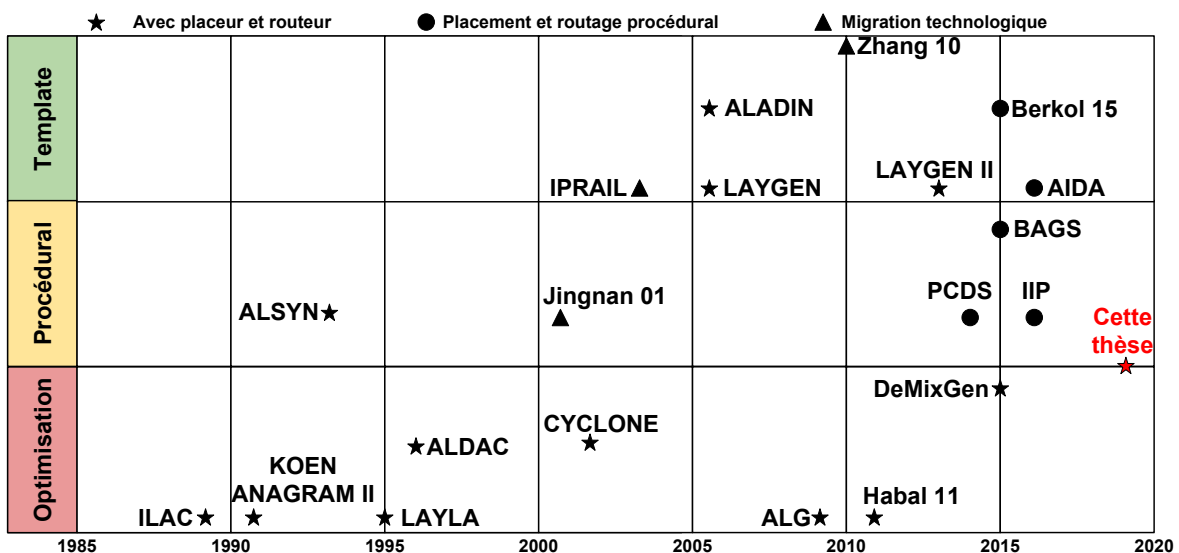


FIGURE 2.12 – Représentation chronologique des outils de CAO dédiés aux dessins des masques analogique entre 1985 et 2018

Le Laboratoire d'Informatique de Paris 6 fait preuve d'une expérience de près d'une trentaine d'années en matières de développement d'outils de CAO dédiés à la conception de circuits numériques et analogiques. Le développement des outils dédiés à la génération des masques de circuits analogiques ont vu le jour à travers plusieurs projets.

L'ensemble des travaux mentionné précédemment sont rassemblés dans le tableau récapitulatif 2.1. Ce tableau présente les outils de CAO analogiques développés au sein du LIP6 en précisant :

- Le nom de l'outil concerné
- La période de développement
- Les thèses et les contributeurs
- La méthode de synthèse électrique utilisée
- Le modèle électrique du transistor MOS
- La grille de représentation du dessin des masques
- L'approche de placement employée
- Les langages d'implémentation

C'est dans la continuité des outils de CAO dédiés à la génération de dessins des masques pour circuits analogiques que cette thèse s'inscrit. La contribution majeure de cette thèse porte sur le placement-routage pour circuits mixtes. Un placeur analogique a été développé dont l'approche est de donner du contrôle au concepteur dans le choix de la topologie et des contraintes du placement. Une phase de routage global permet ensuite d'unifier le flot numérique et le flot analogique en traitant dans une même phase l'ensemble des signaux des deux domaines tout en respectant les contraintes de routage. Une phase de routage détaillé termine le routage en construisant les fils et en résolvant les problèmes de superpositions entre les fils du circuit.

TABLE 2.1 – De Cairo à Coriolis

LOGICIEL Encadrants Contributeurs années	Doctorant Contributeur (MASI et LIP6)	Synthèse électrique	Modèle électrique du transistor MOS	Dessin des masques. Symbolique ou réel	Placement et Routage	Structure de données et Langage de programmation
CAIRO Louërat Greiner 1996-2000	M. Dessouky [42]	externe OCEANE [3]	BSIM3 OCEANE [3]	Grille symbolique. Uniquement analogique.	Placement relatif en tranches. Routage procédural.	ALLIANCE-A C
CAIRO2 Louërat Greiner 2000-2002	P. Nguyen Tuong [43]	externe OCEANE [3]	BSIM3 OCEANE [3]	Grille symbolique. Uniquement analogique.	P. en tranches et Fn de forme. Routage procédural.	ALLIANCE-A C
CAIRO+ Louërat Greiner 2002 - 2008	P. Nguyen Tuong [43] L. de Lamarre [44] V. Bourguet [45] R. Iskander [46]	Flot. Calculette MOS. Opérateurs. Graphe de dépendance. Optimisation.	BSIM3 (intégré)	Placement hybride : symbolique et réel. Uniquement analogique.	Placement relatif en tranches. Fonction de forme. Routage procédural.	ALLIANCE-A C++
CHAMS R. Iskander M.-M. Louërat J.-P. Chaput D. Dupuis 2009-2015	F. Javid [47] S. Youssef [48]	Opérateurs et Graphe biparti. Hiérarchie. Optimisation. Dimensionnement et analyse	Accès aux modèles des simulateurs ex : BSIM3, BSIM4 PSP	P&R au réel. Dispositifs uniquement analogiques. Paramètres de style.	Placement HB-Tree optimisé. Fn de forme. Routage procédural.	HURRICANE-AMS C++ Python
CORLIOLIS M.-M. Louërat J.-P. Chaput J. Porte 2015-2017	E. Lao [49]	OCEANE Hiérarchie. Optimisation. Dimensionnement et performances.	Intégré et accès aux modèles des simulateurs ex : BSIM3, BSIM4	P&R au réel. Circuits mixtes : <i>standard cells</i> numériques et dispositifs analogiques.	Placement en tranches automatisé. Fn de forme. Routage contraint et automatisé.	HURRICANE-AMS C++ Python

2.4. CONCLUSION

Nom de l'outil	Année	Approche	Placement	Routing	Contraintes	Complexité du circuit	Temps d'exécution	Entrées/sorties	Langage	Technologie
ILAC	1989	Optimisation	- Plan de masse: - recuit simulé - Placement: - recuit simulé	- Routing global: - best-first search - Routing détaillé: - channel routing	- appariement de modules - symétries	- 1) voltage reference - 2) operational amplifier (36 transistors) - 3) symmetrical layout of comparator	- (1) 3 min - (2) 3.5 min - (3) 15 min	- Entrées: - fichier SPICE - description layout - netlist - Sorties: - fichier CIF/GDSII	- Pascal	- non mentionnée
KOEN ANAGRAM II	1991	Optimisation	- slicing structure - recuit simulé	- 1 phase: Line-Expansion Routing	- placement symétrique - routage over-the-device - routage symétrique	- 1) CMOS comparator (26 devices) - 2) 2 CMOS op-amp designs (11 devices, 31 devices) - 3) BiCMOS op amp (16 devices)	- placement: - 1-45 min - routage: - 1-45 min	- Entrées: - SPICE netlist - fichier de description de la technologie - Sorties: - fichier format MAGIC	- C	- MOSIS 2-µm p-well CMOS
ALSYN	1993	Procédural	- slicing structure suivant les contraintes du concepteur - optimisation de la slicing structure en termes d'espace	- 1 phase: - exploration exhaustive	- hauteur de la cell - symétrie - centrage géométrique	- 1) 3 Op-amps (17 devices, 54 devices, 71 devices) - 2) 3 Switched capacitor low-pass filters (49 devices, 39 devices, 111 devices)	- (1) 11 sec, 42 sec, 44 sec - (2) 26 sec, 96 sec, 50 sec	- Entrées: - netlist - liste des règles de contraintes - Sorties: - format OCT	- C - C++	- non mentionnée
LAYLA	1995	Optimisation	- non slicing structure, coordonnées absolues - recuit simulé	- non mentionné	- symétrie - appariement de modules - geometrical optimization	- 1) high-speed CMOS comparator - 2) fully differential CMOS operational amplifier - 3) high-speed CMOS operational amplifier	- (1) placement: - 93 sec - (2) placement: - 163 sec - (3) placement: - 83 sec	- Entrées: - netlist - spécifications - Sorties: - non mentionnée	- C++	- non mentionnée
ALDAC	1996	Optimisation	- recuit simulé	- Routing global: - exploration exhaustive - Routing détaillé: - channel routing	- symétrie - appariement de modules	- 1) two-stage CMOS opamp (9 transistors) - 2) clocked comparator (24 transistors) - 3) micro-power amplifier (51 transistors)	- (2) 3602 sec - (3) 2 semaines	- Entrées: - Netlist - spécifications, - Sorties: - non mentionnée	- C - C++	- non mentionnée
Jingnan 01	2001	Procédurale	- PCells; parameterized cell - Migration technologique			- IQDAC	- non mentionné	- Entrées: - netlist, - fichier DRC - Sorties: - GDSII	- SKILL	- FoundryX - 0.35µm à FoundryY - 0.25µm
CYCLONE	2002	Optimisation	- non slicing structure - coordonnées absolues - recuit simulé	- non mentionné	- symétrie - appariement de modules	- VCO	- moins de 24h	- Entrées: - netlist, - spécifications, - fichier DRC - Sorties: - GDSII	- C - C++	- CMOS 0.35µm - BiCMOS 0.65µm
IPRAIL	2003	Template	- Migration technologique - Extraction de template			- (1) Single Ended Folded Cascade Operational Amplifier (43 transistors) - (2) Two-Stage Miller Compensated Operational Amplifier (48 transistors)	- (1) 39 sec - (2) 37 sec	- Entrées: - fichier CIF - fichier de règles des 2 technologies - Sorties: - Fichier CIF	- Non mentionné à TSMC	- TSMC 0.25µm CMOS 0.18µm CMOS
ALADIN*	2006	Template	- Placement global - Placement détaillé	- routage global en se basant sur les minimum-Steiner-tree - algorithme de Lee	- proximité - matching - symétrie - performance	- (1) rail-to-rail Opamp - (2) CMOS Comparator	- (1) placement: - 752 sec - (1) routage: - 1579 sec - (2) placement: - 96 sec - (2) routage: - 943 sec	- Entrées: - netlist - cells - Sorties: - GDSII, CIF	- C++ - SKILL, - TCL/TK	- Thesys-0.8µm CMOS
LAYGEN	2006	Template	- représentation B-Tree - recuit simulé	- utilise un template avec du recuit simulé	- placement relatif - symétries - appariement de modules	- differential amplifier (11 transistors)	- non mentionné	- Entrées: - template d'une base de donnée, - design Kit - Sorties: - GDSII, CIF	- C++ - SKILL - Java	- non mentionnée

FIGURE 2.13 – Description des outils de génération de dessins des masques pour circuits analogiques et mixtes depuis 1985

ALG+	2009	Optimisation	- Placement: - Template+Optimisation - Template donne la position initiale la recherche de solution - partitionnement	- Routage global: - exploration exhaustive - Routage détaillé: - switchbox routing - schéma	- parasité capacitif - appariement de modules - aspect ratio - performance	- Symmetric OTA (12 transistors) - Opamp (19 transistors) - Comparator (14 transistors)	- max 10 min	- Entrées: - spécifications - interactions du concepteur - Sorties: - non mentionné	- Java	- non mentionnée
Zhang 10	2010	Template	- migration technologique			- (1) two-stage Miller-compensated opamp (8 transistors) - (2) single-ended folded cascode opamp (14 transistors)	- (1) 67.5-69.9 sec - (2) 179.8-212 sec	- Entrées: - layout existant - règles de dessin des 2 technologies - Sorties: - non mentionnée	- C++	- 0.25µm CMOS process à 0.18µm CMOS process
Habal 11	2011	Optimisation	- représentation B*-tree - recuit simulé	- recuit simulé	- symétries - proximité - centrage géométrique - routage symétrique - performance	- (1) amplif. (19 transistors) - (2) tunable operational transconductance amplif. (52 transistors)	- (1) 272 min - (2) 665 min	- Entrées: - spécifications - netlist - Sorties: - OpenAccess, XML, LEF DEF, GDSII	- C++	- non mentionnée
LAYGEN II	2013	Template	- représentation B*-tree - recuit simulé	- algorithme génétique	- symétrie - appariement - proximité - routage symétrique - routage des net sensible	- (1) fully-Dynamic Comparator (11 transistors) - (2) single-Ended Folded Cascode Amplifier (11 transistors) - (3) operational Amplifier	- (1) 109 sec - (2) 33 sec - (3) 2h	- Entrées: - spécifications - netlist - Sorties: - GDSII	- Java	- UMC 0.13µm
PCDS	2014	Procédural	- placement relative: choisi par le concepteur - routage: paramétrable			- OTA - OTA with rail-to-rail topology - RC ladder	- non mentionné	- Entrées: - liste de paramètres - Sorties: - GDSII, CIF	- SKILL	- indépendant d'une technol
BAGS	2015	Procédural	- procédural		- appariement de modules - ratio	- (1) VCO - (2) switched capacitor DC-DC regulator	- non mentionné	- Entrées: - template - Sorties: - non mentionné	- Python	- CMOS 65nm - CMOS 40nm
Berkol 15	2015	Template	- procédural		- taille physique des transistors - appariement de modules	- folded cascode OTA (16 transistors)	- 0.3 sec	- Entrées: - spécifications - netlist - Sorties: - fichier GDS	- Matlab	- 130nm CMOS
DemixGen	2015	Optimisation	- architecture donnée par le concepteur	- Routage global: - LLP - Routage détaillé: - channel routing	- symétrie - proximité	- sigma delta modulator (100 transistors)	- 40 sec	- Entrées: - spécifications - SPICE netlist - architecture du circuit - Sorties: - non mentionné	- C++	- TSMC 0.18µm
llp Framework	2016	Procédural	- procédural			- 12 bit current-steering digital-to-analog converter (4095 cascode current mirror stages)	- 110-117 sec	- Entrées: - spécifications - SPICE netlist - architecture du circuit - Sorties: - non mentionné	- Python	- 6 different technologies ranging from 350nm down to 28nm
AIDA*	2016	Optimisation	- à base de template - optimisation du résultat		- topologie - taille physique des transistors - performances - symétrie - matching	- (1) two-stage folded cascode amplif. (19 transistors) - (2) single stage amplif. (12 transistors)	- (1) non mentionné - (2) 2 hours and 45 minutes	- Entrées: - spécifications - netlist - paramètres sous fichier XML - Sorties: - non mentionné	- Java	- 130nm CMOS
Cette thèse	2018	Optimisation	- procédural avec interactions	- Routage global: - algorithme de Dijkstra - Routage détaillé: - rip-up and reoute	- placement: - symétrie - proximité - régularité - bordure - routage: - symétrie	- OTA - Amplificateur à source de courant ajustable - transconductance différentiel contrôlable	- secondes	- Entrées: - Slicing Tree - tailles des modules - nets symétriques - choix de la taille des nets - Sorties: - GDSII	- C++ - Python	- 350nm, 180nm

FIGURE 2.14 – Description des outils de génération de dessins des masques pour circuits analogiques et mixtes depuis 1985(suite)

2.5 Références

- [1] Alain Greiner and François Pêcheux. Alliance : A complete set of CAD tools for teaching vlsi design. 1992. [10](#)
- [2] Christophe Alexandre. *Coriolis : une plate-forme ouverte pour l'évaluation de flots de conception VLSI fortement intégrés*. PhD thesis, Paris 6, 2007. [10](#)
- [3] Jacky Porte. *OCEANE : Outils pour la Conception et l'Enseignement des circuits intégrés ANalogiquEs – reference manual*, January 2018. [10](#), [22](#)
- [4] Anthony Lester, Pirouz Bazargan-Sabet, and Alain Greiner. YAGLE, a second generation functional abstractor for CMOS VLSI circuits. In *Microelectronics, 1998. ICM'98. Proceedings of the Tenth International Conference on*, pages 265–268. IEEE, 1998. [10](#)
- [5] J Kuhn. Analog module generators for silicon compilation. *VLSI Systems Design*, 8(5) :74, 1987. [13](#)
- [6] V Meyer Zu Bexten, C Moraga, R Klinke, W Brockherde, and K-G Hess. Alsyn : Flexible rule-based layout synthesis for analog ic's. *IEEE journal of solid-state circuits*, 28(3) :261–268, 1993. [ix](#), [13](#)
- [7] John Crossley, Alberto Puggelli, H-P Le, B Yang, R Nancollas, Kwangmo Jung, Lingkai Kong, Nathan Narevsky, Yue Lu, Nicholas Sutardja, et al. BAG : A designer-oriented integrated framework for the development of ams circuit generators. In *Proceedings of the International Conference on Computer-Aided Design*, pages 74–81. IEEE Press, 2013. [ix](#), [13](#), [14](#)
- [8] Xu Jingnan, João Vital, and Nuno Horta. A SKILL-based library for retargetable embedded analog cores. In *Proceedings of the conference on Design, automation and test in Europe*, pages 768–769. IEEE Press, 2001. [13](#), [18](#)
- [9] Timothy J Barnes. SKILL : A CAD system extension language. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 266–271. ACM, 1991. [13](#)
- [10] Cadence Design Framework II and IC Version. 5.0. 33, cadence design systems. Inc., San Jose, CA, 2004. [13](#), [15](#)
- [11] Daniel Marolt, Matthias Greif, Jürgen Scheible, and Göran Jerke. PCDS : A new approach for the development of circuit generators in analog IC design. In *Microelectronics (Austrochip), 22nd Austrian Workshop on*, pages 1–6. IEEE, 2014. [14](#), [18](#)
- [12] Benjamin Prautsch, Uwe Eichler, Sunil Rao, Björn Zeugmann, Ajith Puppala, Torsten Reich, and Jens Lienig. IIP framework : A tool for reuse-centric analog circuit design. In *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2016 13th International Conference on*, pages 1–4. IEEE, 2016. [14](#)
- [13] Jef Rijmenants, James B Litsios, Thomas R Schwarz, and Marc GR Degrauwe. ILAC : An automated layout tool for analog CMOS circuits. *IEEE Journal of Solid-State Circuits*, 24(2) :417–425, 1989. [14](#)
- [14] Marc GR Degrauwe, Olivier Nys, Evert Dijkstra, Jef Rijmenants, Serge Bitz, Bernard LAG Goffart, Eric A Vittoz, Stefan Cserveny, Christian Meixenberger, G Van Der Stappen, et al. IDAC : An interactive design tool for analog CMOS circuits. *IEEE Journal of solid-state circuits*, 22(6) :1106–1116, 1987. [14](#)

- [15] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983. [14](#)
- [16] John M Cohn, David J Garrod, Rob A Rutenbar, and L Richard Carley. KOAN/ANAGRAM II : New tools for device-level analog placement and routing. *IEEE Journal of Solid-State Circuits*, 26(3) :330–342, 1991. [14](#)
- [17] Koen Lampaert, Georges Gielen, and Willy Sansen. *Analog layout generation for performance and manufacturability*. Springer US, 1999. [14](#)
- [18] Carl RC De Ranter, Geert Van der Plas, Michiel SJ Steyaert, Georges GE Gielen, and Willy MC Sansen. CYCLONE : Automated design and layout of RF LC-oscillators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(10) :1161–1170, 2002. [14](#)
- [19] Enrico Malavasi, Edoardo Charbon, Eric Felt, and Alberto Sangiovanni-Vincentelli. Automation of IC layout with analog constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(8) :923–942, 1996. [ix](#), [14](#)
- [20] Ender Yilmaz and Günhan Dundar. Analog layout generator for CMOS circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(1) :32–45, Jan 2009. [ix](#), [14](#), [15](#)
- [21] Husni Habal and Helmut Graeb. Constraint-based layout-driven sizing of analog circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(8) :1089–1102, 2011. [15](#)
- [22] Mark Po-Hung Lin, Po-Hsun Chang, Shuenn-Yuh Lee, and Helmut E Graeb. DeMix-Gen : Deterministic mixed-signal layout generation with separated analog and digital signal paths. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(8) :1229–1242, 2016. [15](#)
- [23] Nuttorn Jangkrajarn, Sambuddha Bhattacharya, Roy Hartono, C.-J. Richard, and Shi. IPRAIL – intellectual property reuse-based analog IC layout automation. In *Integration, the VLSI Journal*, number Report, 2003. [15](#)
- [24] Lihong Zhang, Ulrich Kleine, and Yingtao Jiang. An automated design tool for analog layouts. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(8) :881–894, 2006. [ix](#), [15](#), [16](#)
- [25] Ricardo Martins, Nuno Lourenco, and Nuno Horta. LAYGEN II - automatic layout generation of analog integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(11) :1641–1654, 2013. [ix](#), [16](#)
- [26] Gönenç Berkol, Ahmet Unutulmaz, Engin Afacan, Günhan Dünder, Francisco V Fernandez, Ali E Pusane, and F Başkaya. A two-step layout-in-the-loop design automation tool. In *New Circuits and Systems Conference (NEWCAS), 2015 IEEE 13th International*, pages 1–4. IEEE, 2015. [15](#)
- [27] Ahmet Unutulmaz, Günhan Dünder, and Francisco V Fernández. Template coding with LDS and applications of LDS in EDA. *Analog Integrated Circuits and Signal Processing*, 78(1) :137–151, 2014. [15](#)

- [28] Nuno Lourenco, Michele Vianello, Jorge Guilhermel, and Nuno Horta. LAYGEN - automatic layout generation of analog ics from hierarchical template descriptions. In *Research in Microelectronics and Electronics 2006, Ph. D.* IEEE, 2006. 16
- [29] Nuno Lourenço, Ricardo Martins, António Canelas, Ricardo Póvoa, and Nuno Horta. Aida : Layout-aware analog circuit-level sizing with in-loop layout generation. *Integration, the VLSI Journal*, 55 :316–329, 2016. 16
- [30] Synopsys, Inc. <https://www.synopsys.com>. 16
- [31] Cadence Design Systems, Inc. <https://www.cadence.com/>. 16
- [32] Mentor Graphics. <https://www.mentor.com/>. 16, 17
- [33] Synopsys Acquires Ciranova. <https://news.synopsys.com/index.php?s=20295&item=123403>. 16
- [34] Synopsys Completes Acquisition of Magma Design Automation. <https://news.synopsys.com/index.php?s=20295&item=123356>. 16
- [35] Cadence to Acquire Neolinear. https://www.cadence.com/content/cadence-www/global/en_US/home/company/newsroom/press-releases/pr-ir/2004/cadencetoacquireneolinear.html. 16, 18
- [36] Mentor Graphics Acquires Tanner EDA. <https://www.mentor.com/tannereda/place-route>. 16
- [37] Custom Compiler, Synopsys, Inc. <https://www.synopsys.com/implementation-and-signoff/custom-implementation/custom-compiler.html>. 16
- [38] IRoute and ARoute. <http://go.mentor.com/4gsw8>. 17
- [39] Tanner EDA Place and Route. <https://www.mentor.com/company/news/mentor-acquires-tanner-eda>. 17
- [40] Tanner L-Edit IC Layout. <https://www.mentor.com/tannereda/l-edit>. 17
- [41] Cadence PCell Designer. https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/services/cadence-vcad-pcell-ds.pdf. 17
- [42] Mohamed Dessouky. *Design for Reuse of Analog Circuits. Case Study : Very Low-Voltage $\Delta\Sigma$ Modulator*. PhD thesis, Université Pierre et Marie Curie, (UPMC), January 2001. 18, 22
- [43] Pierre Nguyen Tuong. *Définition et implantation d'un langage de conception analogiques réutilisables*. PhD thesis, Université Pierre et Marie Curie, (UPMC), 2006. 18, 22
- [44] Laurent de Lamarre. *CAIRO+ : Intégration du modèle Bsim3v3, stage de DEA*. Université Pierre et Marie Curie, (UPMC), 2002. 18, 22
- [45] Vincent Bourguet. *Conception d'une bibliothèque de composants analogiques pour la synthèse orientée layout*. PhD thesis, Université Pierre et Marie Curie, (UPMC), 2007. 18, 19, 22

- [46] Ramy Iskander. *Knowledge-Aware Synthesis for Analog Integrated Circuit Design and Reuse*. PhD thesis, Université Pierre et Marie Curie, (UPMC), 2008. [18](#), [22](#)
- [47] Farakh Javid. *Synthèse structurée des circuits analogiques intégrés capitalisant la connaissance du concepteur : vers une perspective industrielle*. PhD thesis, Université Pierre et Marie Curie, (UPMC), October 2013. [22](#)
- [48] Stéphanie Youssef. *Designer-assisted, Reusable and Optimized Analog Layout Generation for Nanometric CMOS Era*. PhD thesis, Université Pierre et Marie Curie, (UPMC), December 2012. [22](#)
- [49] Eric Lao. *Placement et routage de circuits mixtes analogiques-numériques CMOS*. PhD thesis, Université Pierre et Marie Curie, (UPMC), July 2018. [22](#)

Chapitre 3

Placement

Sommaire

3.1 Introduction	30
3.2 Formalisation du problème du placement	31
3.2.1 Définition du problème	31
3.2.2 Les contraintes du placement	33
3.3 Les méthodes de résolution existantes	37
3.3.1 Les représentations topologiques de plan de masse	37
3.3.2 Les méthodes d'optimisation de placement	43
3.4 Méthode de résolution	44
3.4.1 Notre approche de placement	45
3.4.2 Définition de la représentation en <i>slicing tree</i>	47
3.4.3 Les nœuds du <i>slicing tree</i> et leurs contraintes	49
3.4.4 L'algorithme de placement	54
3.5 Implémentation du placement	57
3.5.1 Construction du <i>slicing tree</i>	59
3.5.2 Mise en œuvre de l'algorithme de placement	64
3.5.3 Interface utilisateur	68
3.6 Conclusion	71
3.7 Références	72

3.1 Introduction

Les systèmes-sur-puce modernes contiennent aussi bien des circuits numériques que des circuits analogiques. La conception de circuits numériques a été particulièrement automatisée par des outils de conception assistée par ordinateur tandis que la conception de circuits analogiques est restée manuelle. Avec l'évolution des nouvelles technologies nanométriques, cela implique un travail manuel long et sujet à l'erreur humaine. Ainsi, la partie analogique de la génération du dessin des masques de circuits mixtes est la partie limitante du flot de conception mixte.

Considérant les modules d'un circuit mixte, la phase de placement consiste à obtenir une surface de circuit la plus réduite possible et à minimiser la longueur des fils d'interconnexions. De plus, les parties analogiques requièrent de respecter des contraintes supplémentaires afin de réduire les effets parasites des composants causés par les procédés de fabrication. Parmi ces contraintes supplémentaires, on énonce quelques contraintes usuelles :

- **Symétries** : Les contraintes de symétries sont utilisées dans le contexte de circuits différentiels. Le placement symétrique vise à placer des composants dans un même environnement, on souhaite en particulier qu'ils soient soumis aux mêmes effets parasites. L'appariement des composants permet de réduire les effets parasites de par le fonctionnement différentiel (compensation des parasites entre deux signaux).
- **Sens du courant** : Les modules doivent être placés de façon à respecter le sens du courant. On cherche à ce que le courant se propage dans une seule et même direction. Dans le cas contraire, cela peut entraîner des topologies de routage complexes pouvant augmenter les effets parasites ou des longueurs de fil plus longues. Cette contrainte a pour objectif d'apparier au mieux les composants en rendant la structure des fils la plus régulière possible.
- **Proximité** : Cette contrainte impose à des modules de rester relativement proches pour pouvoir, par exemple, partager une même zone de substrat. Elle permet aussi d'éviter les erreurs d'appariement et également de limiter les déviations de processus de fabrication.
- **Modules pré-placés** : Dans certains contextes, il est possible que le concepteur cherche à garder le contrôle total sur le placement de quelques modules sensibles. Le placeur doit alors être capable de placer les modules dans un environnement pouvant être autre que rectangulaire (voir partie 3.2.2).

De manière générale, le problème du placement est différent pour la partie numérique et la partie analogique du circuit. Dans le cas de la partie numérique, le problème consiste à placer un très grand nombre de cellules standard (pouvant être de l'ordre du million cellules) avec peu de contraintes de placement. Pour la partie analogique, le problème consiste à placer un petit nombre (jusqu'à une centaine de modules) avec davantage de contraintes (symétrie, appariement, régularité, ...).

Le placement numérique et le placement analogique sont traités séparément dans notre approche. On estime que ces deux problèmes sont trop différents pour pouvoir être traités par un unique et même algorithme. Pour cela, le placement numérique est réalisé en utilisant le placeur numérique de Coriolis[1]. Il est possible d'utiliser ensuite le résultat

du placement en tant que module pour le placement des modules analogiques.

La qualité du placement a un impact conséquent sur la phase du routage et donc sur les performances d'un circuit. La capacité à pouvoir gérer au mieux des contraintes analogiques est considérée par les différentes méthodes de résolution de l'état de l'art. Dans le cadre de cette thèse, le placeur analogique a été développé dans son intégralité, un placeur numérique est déjà présent dans **Coriolis**.

Dans ce chapitre, nous présentons dans un premier temps le problème du placement analogique et mixte avec sa représentation et les différentes contraintes de placement employées. Dans un second temps, nous présentons les méthodes de résolution usuelles de l'état de l'art, en particulier les différentes représentations employées. Ensuite, nous expliquons notre méthode de résolution avec une approche semi-automatique faisant intervenir les choix du concepteur ainsi les détails de l'algorithme de placement mixte. Nous décrivons par la suite la mise en œuvre et la structure de données utilisées. Enfin, la description de l'utilisation des scripts pour générer le placement et l'interface utilisateur seront présentées.

3.2 Formalisation du problème du placement

La résolution du problème du placement analogique et mixte se base sur une représentation du plan de masse. Elle permet de déterminer la position de chacun des modules du plan ainsi que l'ensemble des contraintes appliquées. Il existe de nombreuses représentations topologiques et chacune possède ses règles de placement. Dans cette section, on présente le problème du placement analogique et mixte ainsi que les différentes contraintes de placement considérées pour les modules analogiques.

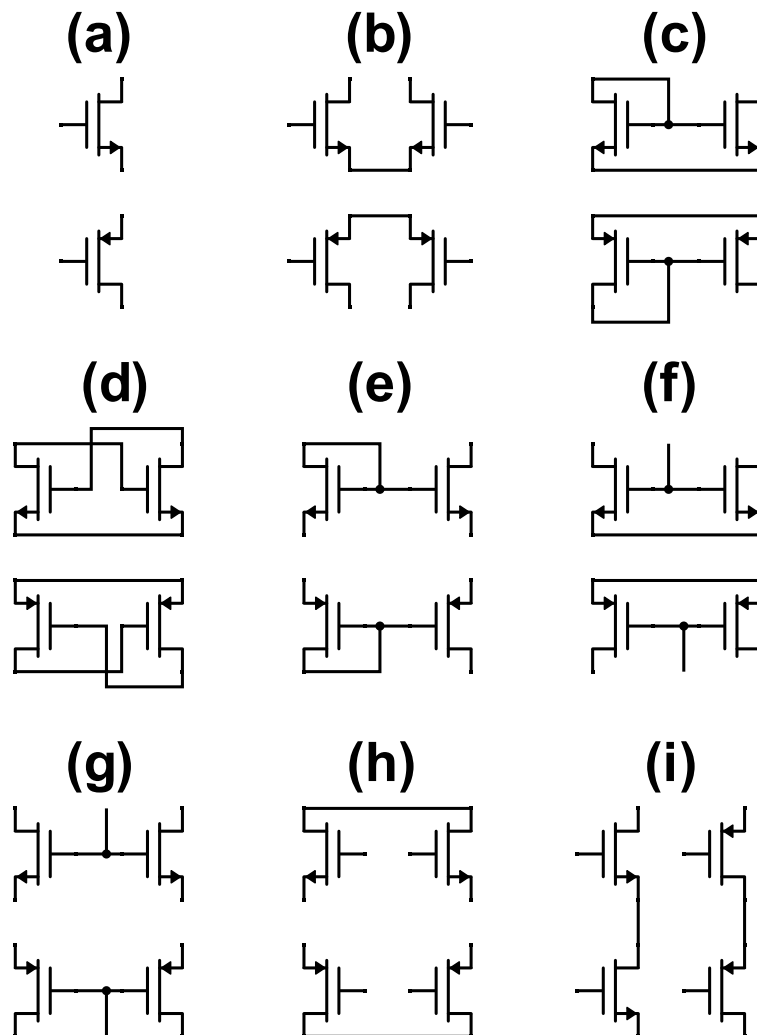
3.2.1 Définition du problème

Suite à la phase de dimensionnement de la partie analogique du circuit, le dessin des masques des modules analogiques peut être généré à partir des informations géométriques (hauteur et largeur de transistor).

Définition : On appelle un module analogique, un ensemble de transistors réalisant une fonction atomique analogique. En particulier, dans le cadre de nos outils, un module peut être un des éléments suivants :

- **(a) Un transistor simple** : Ce module est composé de quatre connecteurs (un drain, une source, une grille et un substrat)
- **(b) Une paire différentielle** : Ce module est composé de six connecteurs (deux drains, une source, deux grilles et un substrat).
- **(c) Un miroir de courant** : Ce module est composé de quatre connecteurs (deux drains, une source et un substrat).
- **(d) Une paire croisée** : Ce module est composé de quatre connecteurs (deux drains, une source et un substrat).
- **(e) Un décaleur de tension** : Ce module est composé de quatre connecteurs (un drain, deux sources et un substrat).
- **(f) Un montage source commune** : Ce module est composé de cinq connecteurs (deux drains, une source, une grille et un substrat).

- **(g) Un montage grille commune** : Ce module est composé de six connecteurs (deux drains, deux sources, une grille et un substrat).
- **(h) Un montage drain commun** : Ce module est composé de six connecteurs (un drain, deux sources, deux grilles et un substrat).
- **(i) Un montage cascade** : Ce module est composé de six connecteurs (un drain, une source, deux grilles et un substrat).

FIGURE 3.1 – Ensemble des modules générables par **Coriolis**[1]

Chacun de ces modules est configurable par un ensemble de paramètres contrôlant les dimensions géométriques et les aspects électriques des modules. Ces modules développés par Stéphanie Youssef[2] font partie de **Coriolis**[1] et sont les modules considérés pour la phase de placement et de routage dans le cadre de notre approche. Concernant les approches de l'état de l'art, la notion de module reste relativement similaire en termes de complexité du module.

Connaissant les dimensions géométriques des modules d'un circuit, le problème du placement consiste à placer l'ensemble des modules suivant la liste des interconnexions (*netlist*). Plusieurs critères permettent d'évaluer la qualité d'un placement et certains d'entre

eux dépendent du contexte, autrement dit du type de circuit analogique.

Soit un circuit C composé de m modules avec $m \in \mathbb{N}$, on note "Surface $_i$ " la surface occupée par un module $i \in \{1, \dots, |m|\}$ et "Surface(espaces blancs)" la surface des espaces blancs du circuit. On appelle "Performance $_{dégradation}$ " la dégradation de performances par rapport au cas théorique noté "Performance $_{théorique}$ ". Le critère "Longueur de fils" définit la longueur totale des fils d'interconnexions des n nets, avec $n \in \mathbb{N}$, du circuit noté net_j avec $j \in \{1, \dots, |n|\}$, une solution de placement est représentée par :

$$\begin{aligned} Surface(C) &= \left(\sum_{i=0}^m Surface_i \right) + Surface(\text{espaces blancs}) \\ Longueur\ de\ fils_{total}(C) &= \sum_{j=0}^n Longueur\ de\ fils_{net_j} \\ Performances_{réel}(C) &= Performances_{théorique}(C) - dégradations \end{aligned} \quad (3.1)$$

De manière générale, on cherche à minimiser la surface du circuit et la longueur totale des fils d'interconnexions. Une augmentation de la surface du circuit entraîne une augmentation de la longueur des fils. Les fils de routage induisent des effets parasites liés au dessin des masques ce qu'on cherche donc à minimiser. Ces deux facteurs impactent les performances du circuit, il existera toujours une dégradation par rapport au cas théorique, on cherche donc à obtenir les meilleures performances possibles.

Afin d'obtenir le meilleur résultat possible pour chacun de ces critères, il est nécessaire de respecter des contraintes de placement visant à réduire au mieux les effets parasites liés au dessin des masques. Ces contraintes de placement sont des contraintes généralement spécifiques aux parties analogiques. La partie suivante décrit les différentes contraintes à considérer permettant d'obtenir la meilleure solution de placement de la partie analogique.

3.2.2 Les contraintes du placement

Le résultat du placement analogique impacte fortement les performances d'un circuit, qu'il soit numérique ou analogique. Plus particulièrement, le respect de contraintes de placement permet de réduire les effets parasites induits par le dessin des masques. Dans le cadre des circuits mixtes, le placement analogique est plus contraignant que le placement numérique et nécessite davantage d'attention. Cette partie liste l'ensemble des contraintes de placement analogique les plus courantes.

Contraintes de symétries

Les contraintes de symétries sont les contraintes topologiques les plus communes et consistent à placer un ou plusieurs modules de manière symétrique. Elles ont pour effet de diminuer les erreurs d'appariement induites par les procédés de fabrication. Les dimensions géométriques et la structure interne des modules sont identiques entre deux modules symétriques. Il existe différentes symétries comme les symétries de modules selon un point, appelé centrage géométrique (voir figure 3.2.(a)), ou selon un axe de symétrie (voir figure 3.2.(b)).

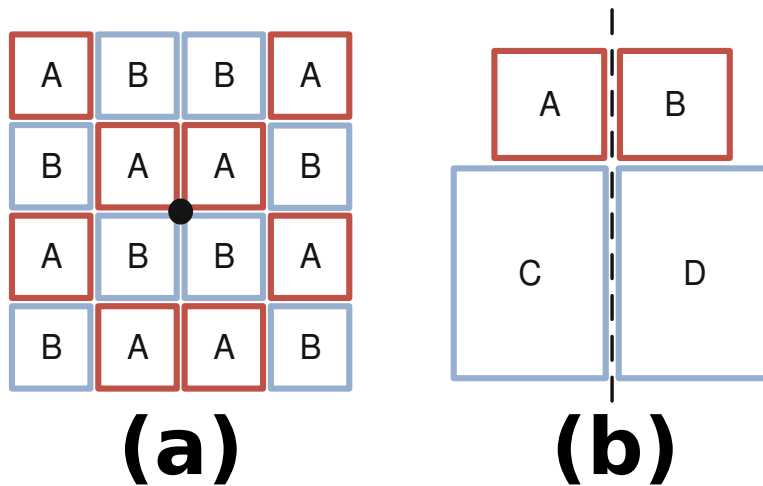


FIGURE 3.2 – Contraintes de centrage géométrique (a) et de symétries (b)[3]

Contraintes du sens du courant

De manière générale, il est préférable de minimiser les effets parasites pouvant affecter des fils de signaux critiques. Pour cela, les modules d'un même *net* critique doivent être proches les uns des autres tout en conservant le sens de la propagation du courant. Sur la figure 3.15.(a), le module M_{11} n'est pas placé de telle sorte à ce que la propagation du sens du courant se fasse dans une seule direction dégradant ainsi les performances du circuit par rapport à la figure 3.15.(b).

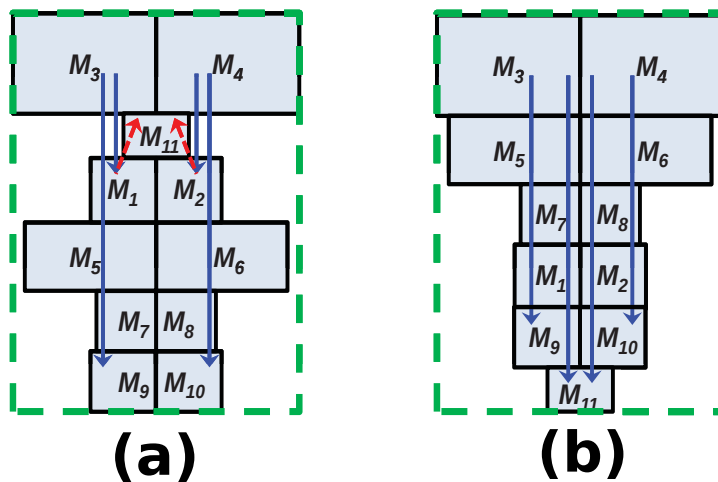


FIGURE 3.3 – Placement sans (a) et avec (b) une prise en compte du sens du courant[4]

Contraintes de température

Les effets thermiques dégagés par les modules du circuit peuvent engendrer une dégradation des performances du circuit en cas de placement inapproprié. En particulier, les modules de puissance dégagent de la chaleur pouvant affecter le fonctionnement électrique des modules sensibles à proximité. Pour des raisons d'appariement, il est préférable de placer des modules sensibles appariés de manière symétrique par rapport aux

modules de puissance afin qu'ils fonctionnent à la même température. La figure 3.4 présente différentes répartitions de modules de puissance et la représentation du dégagement de chaleur induit.

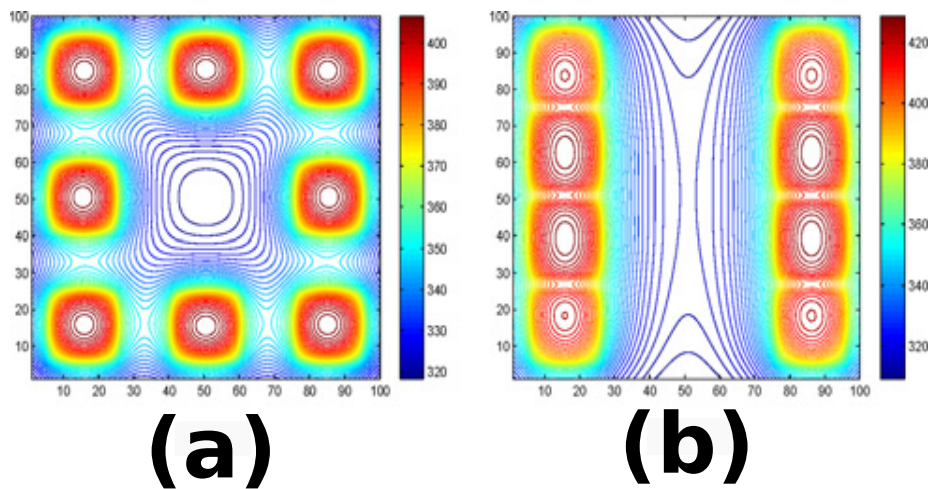


FIGURE 3.4 – Profil thermique du circuit lorsque tous les modules sont répartis équitablement entre les quatre côtés (a) et deux côtés (b) du circuit[5]

Contraintes de proximité

Une contrainte de proximité consiste à restreindre la distance séparant un ensemble de modules. L'objectif des contraintes de proximité est d'améliorer l'appariement entre modules, de réduire la longueur des fils entre les modules contraints, ou bien de faire en sorte de pouvoir partager un substrat/*well* entouré d'un anneau de garde (voir figure 3.5). Des contraintes de proximité peuvent également se traduire par une organisation hiérarchique des modules du circuit.

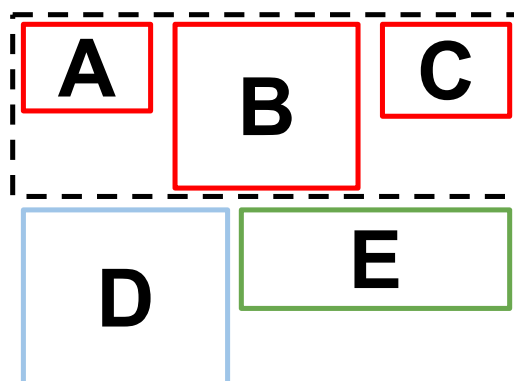


FIGURE 3.5 – Modules soumis à une contrainte de proximité (en rouge) pouvant être entourés d'un anneau de garde[3]

Contraintes de régularité

Les contraintes de régularité sont de plus en plus importantes pour la réalisation du dessin des masques analogiques avec les technologies avancées. D'après Shigetoshi Nakatake[6][7]

et Pang-Yen Chou[8], des structures régulières à tout niveau hiérarchique, que ce soit pour des modules ou ensemble de modules, garantissent de meilleurs résultats en termes de dessin des masques compacts, une meilleure routabilité et moins de sensibilité vis-à-vis des influences des procédés de fabrication. Les figures 3.6 et 3.7 présentent un exemple dans lequel la régularité peut permettre un meilleur routage.

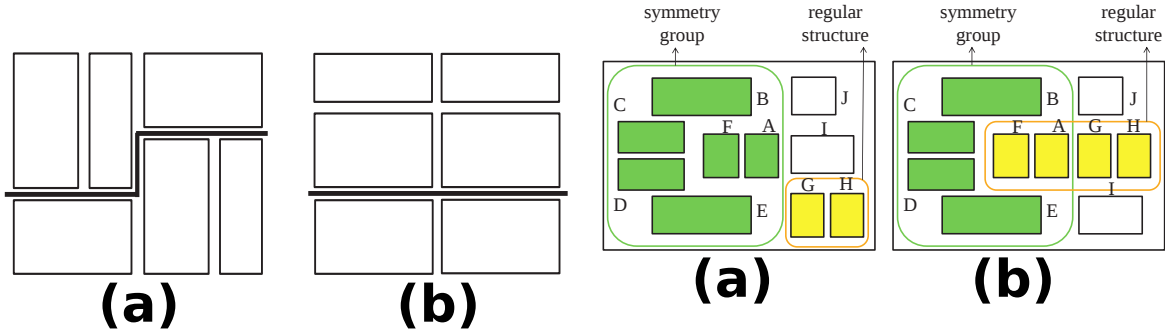


FIGURE 3.6 – (a) Placement sans prise en compte de la régularité (b) Placement avec prise en compte de la régularité[8]

FIGURE 3.7 – (a) Placement privilégiant des contraintes de symétries plutôt que des contraintes de régularité. (b) Placement tenant compte des contraintes de symétries et de régularité en même temps.[8]

Contraintes de modules pré-placés

Il est possible que le concepteur souhaite placer manuellement certains modules afin d’avoir une topologie particulière ou bien lorsque le concepteur souhaite placer des modules à un endroit précis du circuit. Le plan de masse disponible pour le placement des autres modules devient alors irrégulier (originellement un rectangle ou un carré). La figure 3.7 présente un exemple dans lequel trois modules A, B et C sont pré-placés.

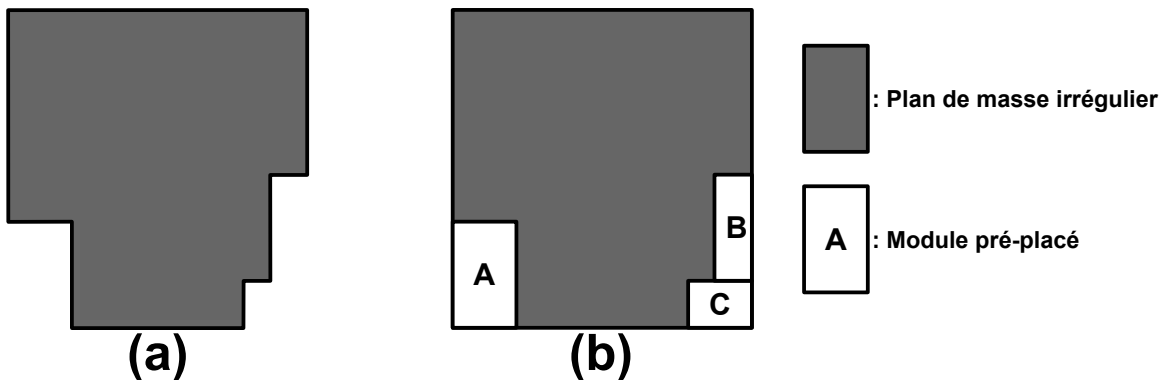


FIGURE 3.8 – Plan de masse irrégulier[9]

Contraintes de bordures

On définit une contrainte de bordure comme étant une contrainte restreignant des modules à être placés uniquement au niveau des bordures d’une région rectangulaire définie par un groupe de modules. Selon Cheng-Wu Lin et al.[10], les contraintes de bordures permettent de limiter la longueur des fils, en particulier entre un groupe de modules et ses connexions extérieures, ce qui a pour conséquence de limiter les effets parasites liés aux fils de routage.

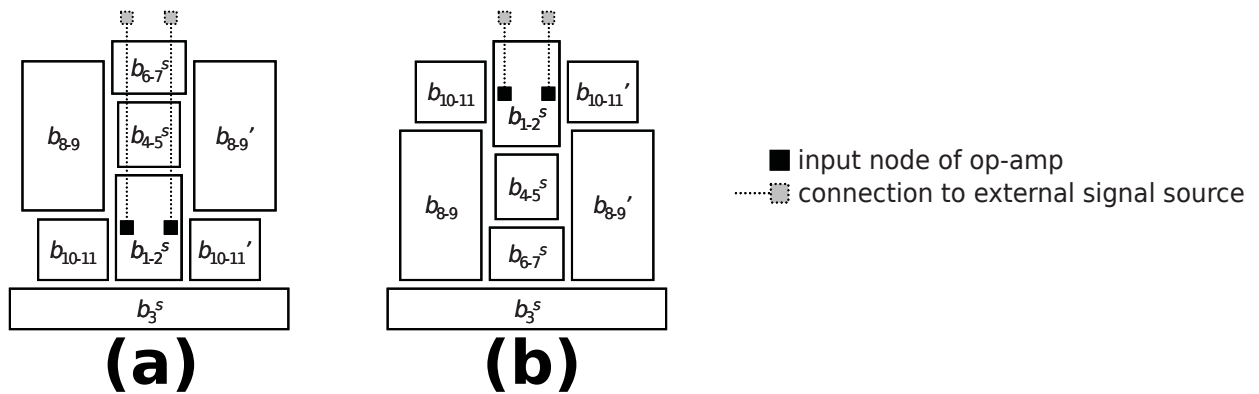


FIGURE 3.9 – Deux exemples de placement d’un amplificateur opérationnel. (a) Les ports d’entrées sont placés au centre du groupe de modules symétriques (b) Les ports d’entrées sont placés en bordure du groupe de modules symétriques [10]

3.3 Les méthodes de résolution existantes

La résolution du problème de placement abordé par les méthodes de résolutions existantes consiste à explorer un grand espace de solutions de placement réalisables ou non réalisables en utilisant une représentation du plan de masse couplée à une optimisation stochastique telle que l’algorithme de recuit simulé. A la différence des circuits numériques, les circuits analogiques doivent prendre en considération des contraintes de placement supplémentaires liées aux parasites engendrés par le dessin des masques. L’état de l’art des approches de placement pour circuits analogiques et mixtes sera abordé dans les sections suivantes.

3.3.1 Les représentations topologiques de plan de masse

Afin de pouvoir générer un placement valide respectant plusieurs contraintes de placement analogiques, la majorité des études récentes emploie une représentation sous forme de graphe. On distingue deux catégories de représentations : les représentations absolues et les représentations topologiques. Les représentations absolues[11] sont utilisées dans les anciennes méthodes de placement et consistent à associer à chacun des modules du circuit une position par rapport à un point de référence. Cette représentation permet la superposition illégale de modules lors de la phase d’optimisation car il n’existe pas de relation de placement entre les modules. Cette approche doit alors être en mesure d’explorer un très grand espace de solutions comprenant aussi bien des placements réalisables que non réalisables. Cela se traduit par des temps d’exécution longs causés par le grand nombre de mouvements nécessaires pour obtenir un dessin des masques satisfaisant. Il est également possible qu’en plus du long temps d’exécution, cette approche ne garantisse pas toujours de solutions réalisables. L’ajustement de la fonction de coût pour éviter les recouvrements de modules peut requérir un effort conséquent.

Les représentations topologiques visent à résoudre les problèmes rencontrés par les représentations absolues comme par exemple les problèmes de recouvrements. Ces représentations topologiques consistent à définir des positions relatives entre les modules d’un circuit. Ces représentations sont largement utilisées pour des raisons d’efficacité et de flexibilité à pouvoir satisfaire des contraintes pour un moindre coût en termes de nombre de mouvements comparées aux représentations absolues. Dans les sous-sections

suivantes, nous décrivons les représentations topologiques les plus utilisées de l'état de l'art.

Les placements en tranche

Les placements en tranche de D. F. Wong et C. L. Liu[12] font partie des premières représentations topologiques utilisées. Ces placements en tranche sont représentés à l'aide d'un graphe appelé *slicing tree*. Un *slicing tree* est un arbre dans lequel chaque nœud interne peut être obtenu par découpes récursives du circuit (voir figure 3.10). À partir de la région totale du circuit représentant la racine de l'arbre, le circuit sera découpé de manière hiérarchique, alternativement de manière horizontale et verticale, jusqu'à atteindre les feuilles de l'arbre représentant les modules du circuit ou bien des espaces de routage.

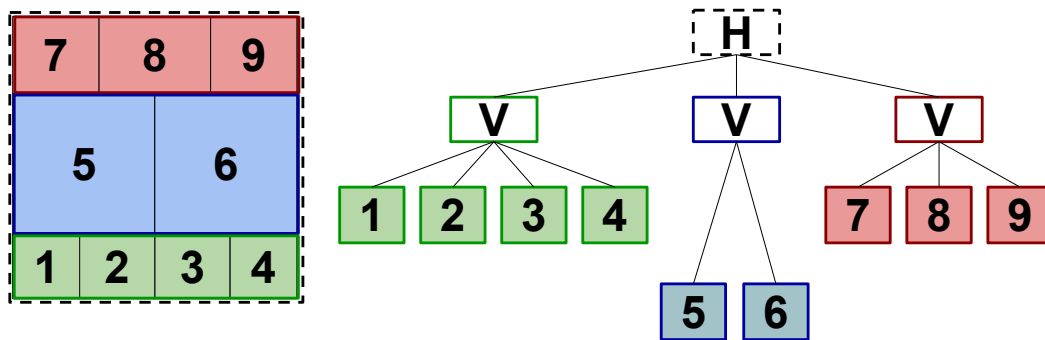


FIGURE 3.10 – Exemple de placement et sa représentation en *slicing tree* où "H" définit une coupe horizontale et "V" une coupe verticale.

T. Abthoff et F. Johannes[13] concentrent leur optimisation sur la surface totale du circuit. De plus, ils prennent en compte la longueur de fils, les contraintes de proximité, de modules pré-placés et de symétries. Juan A. Prieto et al.[14] considèrent les parasites d'interconnexions dans la phase d'optimisation dans une étape simultanée du placement et du routage. Ils sont en mesure de maintenir des symétries de groupes de modules. F.Y. Young et al.[9] étendent l'approche de D. F. Wong et C. L. Liu[12] afin de pouvoir considérer des modules pré-placés, des contraintes de bordure[15] et des contraintes de proximité[16]. [17] améliore la formulation des *slicing trees* en introduisant des conditions de symétries pour des groupes de modules. Po-Hsun Wu et al.[4] introduisent la prise en compte des contraintes de sens du courant dans les structures en *slicing tree* en plus des symétries.

Tout circuit ne peut être représenté par une structure en tranche. De plus, le désavantage de cette topologie est que la densité de la solution de placement peut être dégradée d'une manière notable lorsque les modules d'un circuit ont des rapports hauteur/largeur très hétérogènes. De plus, la structure des *slicing trees* ne permet pas de représenter tous les placement possibles (voir figure 3.11). Suite à ces limitations, des représentations avec des structures *non-slicing* se sont succédé. Les *sequence pairs*, les *B*-trees*, les *Ordered-Trees - O-tree* et les *Transitive Closure Graphs - TCG* seront présentés dans les sous-sections suivantes.

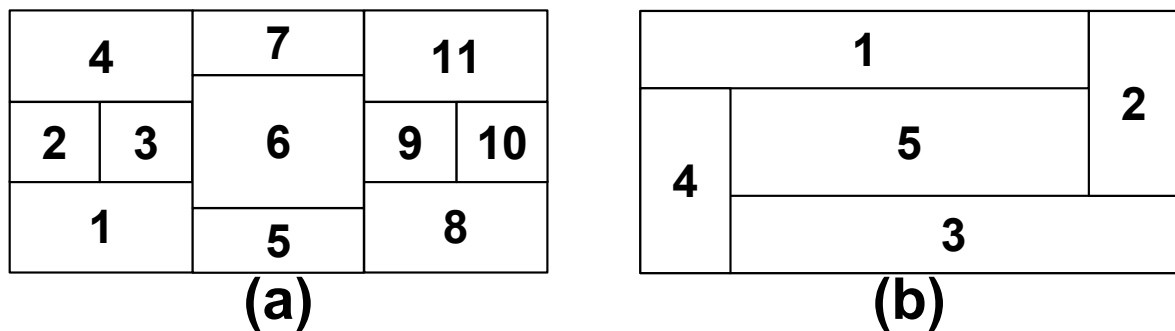


FIGURE 3.11 – Exemple d'un placement représentable par un *slicing tree* (a) et d'un placement non représentable (b)

Les sequence pairs

Les *sequence pairs* sont utilisées dans le contexte des placements de circuits analogiques pour la première fois par Hiroshi Murata et al. [18]. Une *sequence pair* est une paire de séquences/listes ordonnées représentant les positions relatives de l'ensemble des modules. Par exemple, " (abc, cba) " est une *sequence pair* pour un circuit constitué de modules $\{a, b, c\}$. Pour un placement donné, la *sequence pair* correspondante est obtenue par la construction d'une séquence d'échelons positifs et d'une séquence d'échelons négatifs de chacun des modules (voir figure 3.13).

La séquence de l'échelon positif d'un module contient 3 parties : l'échelon haut-droit, l'échelon bas-gauche et la ligne diagonale effectuant la connexion. (voir figure 3.12).

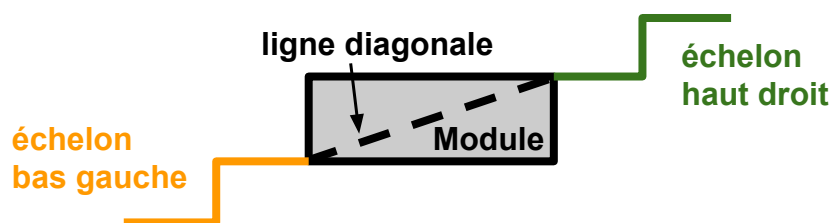


FIGURE 3.12 – Description d'un échelon positif pour un module

L'échelon haut-droit d'un module est dessiné en suivant les règles suivantes :

1. On part du coin haut droit du module.
2. On change la direction alternativement haut et droit jusqu'à atteindre le coin supérieur droit du circuit placé sans croiser les bordures des autres modules et les lignes précédemment tracées.

L'échelon bas-gauche d'un module est dessiné en suivant les règles suivantes :

1. On part du coin bas gauche du module.
2. On change la direction alternativement bas et gauche jusqu'à atteindre le coin inférieur gauche du circuit placé sans croiser les bordures des autres modules et les lignes précédemment tracées.

L'ordre des séquences des échelons positifs correspond aux tracés allant de la gauche vers la droite. On utilise le raisonnement analogue pour les échelons négatifs. À partir d'une paire de séquence d'échelons positifs et négatifs (Γ_+, Γ_-), un placement peut être obtenu. La figure 3.13 montre le placement résultant de ses échelons négatifs et positifs avec ses séquences respectives. A partir d'une *sequence pair*, il est également possible de dégager les contraintes géométriques (horizontale ou verticale) entre deux modules.

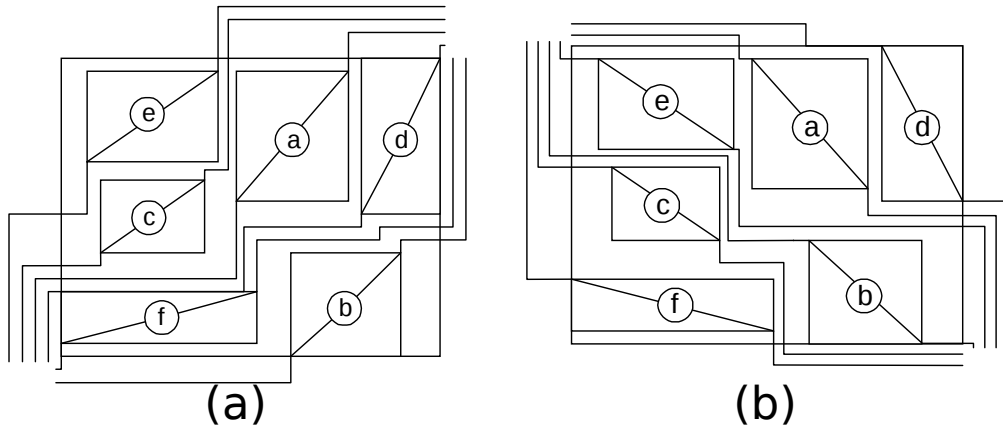


FIGURE 3.13 – (a) Échelons positifs résultant et $\Gamma_+ = ecadfb$. (b) Échelons négatifs résultant et $\Gamma_- = fcbead$. [18]

F. Balasa et K. Lampaert[19] introduisent l'utilisation des *sequence pairs* pour le placement de circuits analogiques avec une prise en compte de contraintes de symétries et F. Balasa et Sarat C Maruvada[20] améliorent leur temps de calcul par la suite. Yiu-Cheong Tam et al.[21] ajoutent la considération d'alignement et de contraintes de proximités des modules en plus des symétries. Long Di et al.[22] apportent une attention particulière à la prise en compte du sens du courant tout en respectant des contraintes de symétries. Shigetoshi Nakatake et al.[6][7] et Pang-Yen Chou et al.[8] prennent en compte des contraintes de régularité couplées aux contraintes de symétries. Linfu Xiao et al.[23] prennent en considération des contraintes de congestions pour le routage en évaluant la congestion des placements donnés.

Les B^* -trees et HB^* -trees

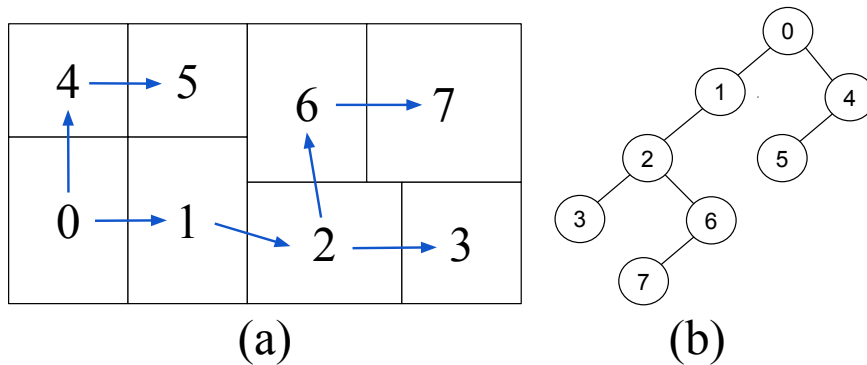


FIGURE 3.14 – (a) Placement compact. (b) Représentation en B^* -tree du placement compact (a).

Les B^* -trees sont des arbres couramment utilisés pour représenter un placement compact de modules dans lequel tous les modules ne peuvent plus se déplacer ni vers la gauche ni vers le bas. Chaque module d'un circuit est représenté par un nœud dans un B^* -tree. La racine d'un B^* -tree correspond au module situé dans le coin inférieur gauche, il s'agit du module 0 sur la figure 3.14. Le nœud de gauche du module 0 représente le module de droite et adjacent au module 0, il s'agit du module 1. Le nœud de droite du module 1 représente le module au dessus et adjacent au module 0, il s'agit du module 4. La figure 3.14 montre un placement et sa représentation en B^* -tree correspondante.

Les B^* -trees ont été introduits par F. Balasa [24] et Yun-Chih Chang et al.[25]. Cette approche est améliorée en incorporant les contraintes de symétrie par F. Balasa et al.[26] dans le cas des circuits analogiques. Sarat C Maruvada et al.[27] améliorent le temps d'exécution de 20% à 30%. Martin Strasser et al.[28] proposent une approche déterministe permettant une meilleure reproductibilité du placement.

La représentation des B^* -tree s'enrichit avec l'utilisation des *Hierarchical Ordered Binary Trees - HB*-trees* de Mark Po-Hung Lin et al.[29] dans le but d'ajouter des contraintes hiérarchiques et groupes de symétries. Cheng-Wu Lin et al.[10] y ajoutent la prise en compte de contraintes de bordure. Cette représentation est ensuite utilisée pour différentes contraintes d'optimisation, telles que des contraintes de température par Mark Po-Hung Lin et al.[5] ou de régularité par Pang-Yen Chou et al.[8] et avec une meilleure optimisation du temps d'exécution par Hui-Fang Tsao et al.[30].

Les O-trees

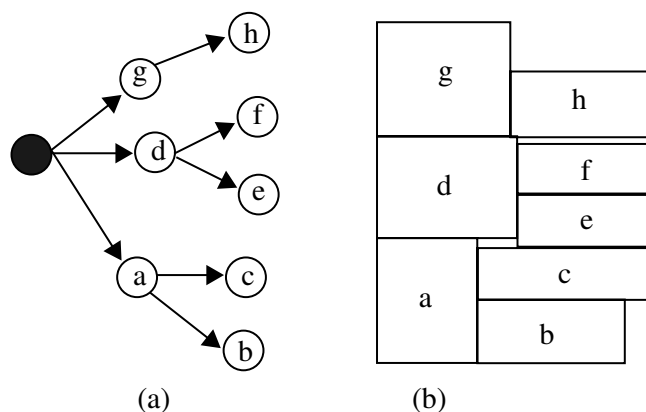


FIGURE 3.15 – (a) Une représentation en O -tree et (b) son placement correspondant. Pour ce placement de 8 modules, on a $T=0010110010110011$ et $\pi=abcdefgh$.

Un *Ordered Tree (O-tree)* étend le principe des B^* -tree. Étant donné un placement de n modules, un O -tree correspondant possède $n + 1$ nœuds encodé par (T, π) , où T correspond à une chaîne de caractères de $2n$ -bit identifiant la structure de l'arbre et π représente l'ordre des noms des modules sans considérer la racine de l'arbre. Lorsqu'on parcourt l'arbre en profondeur, "0" correspondant à une arête descendante, et "1" à une arête ascendante. Un O -tree, dans lequel les nœuds représentent un module rectangulaire, impose des contraintes de positionnement horizontales et verticales :

- (a) Chaque nœud parent doit être placé à la gauche de ses nœuds fils.
- (b) Si deux modules possèdent la même coordonnée en abscisse, le nœud ayant l'index le plus élevé dans l'ordre π doit être placé au-dessus du module ayant un index plus petit.

La figure 3.15 présente un exemple d'*O-tree* composé de huit modules dont le placement est encodé avec $T = 0010110010110011$ et $\pi = abcdefgh$. Cette méthode permet de réduire les redondances et le temps d'exécution est plus court que celui des représentations en *sequence pair*.

Cette méthodologie est introduite par Pei-Ning Guo et al.[31], Yingxin Pang et al.[32] et Linfu Xiao et al.[33] élargies les types contraintes de symétries considérées.

Les Transitive Closure Graph

Les *Transitive Closure Graphs (TCG)* décrivent les relations géométriques entre les modules d'un circuit en se basant sur deux graphes : un *Horizontal transitive closure graph* G_h et un *Vertical transitive closure graph* G_v . Dans le G_h (respectivement G_v), une arête $\langle v_i, v_j \rangle$ exprime que le module m_i se trouve à gauche de (respectivement en bas de) la cellule m_j . Le poids associé à l'arête dans G_h (respectivement G_v) correspond à la largeur (respectivement hauteur) du module associé. La figure 3.16 montre un exemple de placement avec sa représentation *TCG*.

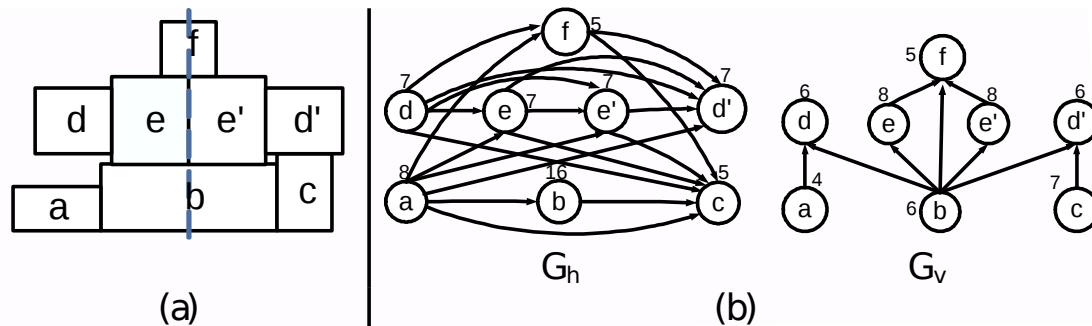


FIGURE 3.16 – (a) Placement compact. (b) Représentation en *TCG* du placement compact (a).

Les *Transitive Closure Graphs (TCG)* ont été introduits par Lin Jai-Ming et Yao-Wen Chang[34] afin de proposer des solutions différentes des *sequence pairs* et des *B*-trees*. Lihong Zhang et al.[35] y ajoutent la considération des contraintes de symétrie.

Tableaux des contraintes des placeurs de l'état de l'art

Au cours de ces deux dernières décennies, de nombreuses représentations topologiques ont été utilisées en couplage avec diverses contraintes. L'ensemble des articles mentionnés précédemment sont rassemblés dans le tableau récapitulatif 3.1. Ce tableau présente la capacité à prendre en compte des contraintes de placement pour chacune des approches citées précédemment.

TABLE 3.1 – Tableau récapitulatif de l'état de l'art des représentations de placement pour les circuits analogiques

Contraintes	Slicing-tree	Sequence Pair	O-tree	B*-tree et HB*-tree	TCG
Symétries	[13], [14], [17], [4]	[19], [20], [21], [7], [23]	[32], [33]	[24], [26], [27], [28], [29], [10], [5], [8], [30]	[35], [36]
Symétrie de groupes	[4]	[23]		[29], [10], [5]	
Centrage géométrique		[23]	[33]	[28]	
Sens du courant	[4]	[22]			
Température				[5]	
Proximité	[13], [16]	[21]		[28]	
Régularité				[24]	
Modules pré-placés	[13], [9]			[30]	
Bordures	[15]			[10], [30]	
Routage	[14]	[23]			

3.3.2 Les méthodes d'optimisation de placement

La majeure partie des approches de l'état de l'art utilise une représentation topologique parmi celles mentionnées dans la section précédente, sur laquelle on applique l'algorithme du recuit simulé. Cette méthode introduite par Scott Kirkpatrick et al. [37] est une technique probabiliste d'approximation de l'optimum global d'une fonction donnée. C'est à dire qu'il s'agit d'une métaheuristique permettant d'approcher l'optimisation globale dans un grand espace de recherche. Le principe s'inspire du processus de recuit des métaux en métallurgie dans lequel le refroidissement d'un matériau est contrôlé. Dans l'application d'une recherche de solution de placement, la température (initialement haute) implique que l'algorithme a une forte chance d'accepter une perturbation de la représentation. Lorsque la température devient basse, les chances de perturbations de la représentation deviennent plus faibles. La probabilité d'acceptation est dégressive tout au long de l'algorithme.

Le fonctionnement de l'algorithme du recuit simulé se déroule suivant les étapes suivantes :

1. L'algorithme sélectionne une transformation aléatoire prédéfinie sur la solution courante.
2. On mesure la qualité de la solution sélectionnée.
3. On sauvegarde la nouvelle solution ou bien on décide de maintenir la solution courante en se basant sur une probabilité dépendant de la qualité précédemment évaluée et de la température.
4. Le paramètre de température diminue se traduisant par une augmentation de la probabilité du choix de la meilleure solution et une diminution de la probabilité du choix de mauvaise qualité.

Le tirage au sort (étape 1) des transformations est déterminé en fonction de l'approche choisie. Ces transformations dépendent de la représentation utilisée, une transformation revient à perturber le graphe représentant le placement de la solution courante. Il peut s'agir par exemple d'un échange de position entre deux modules.

La spécification des contraintes de placement d'un concepteur s'exprime par la formulation d'une fonction de coût qui nous permet d'évaluer la qualité (étape 2) de la solution sélectionnée. Afin d'illustrer le fonctionnement général d'une fonction de coût, considérons une optimisation sur une contrainte de surface globale et d'une longueur de fil, la fonction de coût se présenterait de la façon suivante :

$$Cost(F) = \alpha.A_{normalisé} + \beta.W_{normalisé} \quad (3.2)$$

avec α et β étant des paramètres de réglages permettant d'influencer l'importance de la contrainte considérée, $A_{normalisé}$ représentant la surface totale occupée normalisée et $W_{normalisé}$ la longueur de fil totale normalisée.

3.4 Méthode de résolution

Parmi l'ensemble des méthodologies de placement analogique, la majorité d'entre elles produit un placement en utilisant l'algorithme de recuit simulé et en tenant compte d'une ou plusieurs contraintes du concepteur. Néanmoins, la complexité des contraintes des circuits analogiques et mixtes peuvent rendre difficile l'ajustement des paramètres de la fonction de coût.

Pour rappel de la partie 3.2.1, la qualité d'un placement analogique dépend de plusieurs critères :

- **Surface d'occupation** : Ce critère est commun aux phases de placement numérique et de placement analogique. Pour des raisons de coût et de performances, le résultat du placement doit occuper le moins de surface possible.
- **Capacité d'ajustement** : Il est usuel pour les concepteurs de vouloir effectuer des ajustements sur le résultat d'un placement automatisé. Le placeur analogique doit être suffisamment paramétrable pour pouvoir permettre au concepteur d'ajuster le résultat du placement.
- **Respect de contraintes de placement** : La capacité à placer des modules analogiques tout en respectant des contraintes permet la réduction d'effets parasites liés au dessin des masques pouvant affecter les performances du circuit. Il est nécessaire d'être capable de respecter plusieurs contraintes de placement.
- **Temps d'exécution** : Le temps d'exécution se doit d'être raisonnable. On entend par raisonnable un temps d'exécution inférieur à quelques heures. Un placement automatisé requiert souvent plusieurs exécutions pour des ajustements du placement et des contraintes.
- **Routabilité** : La surface d'occupation se doit d'être la plus petite possible mais elle doit permettre également à la phase de routage d'avoir suffisamment de place pour placer les fils de routage. La phase de placement doit être réalisée en considérant également la phase de routage qui la suit.

Un placeur automatisé doit être capable de répondre à l'ensemble de ces critères. On remarque que dans l'ensemble des placeurs analogiques de l'état de l'art, les considérations de routabilité et de contrôle du concepteur sur le résultat du placement sont souvent négligées. Dans la section suivante, nous présentons l'approche choisie pour notre placeur analogique et mixte ainsi que les raisons qui nous ont poussés à faire ces choix.

3.4.1 Notre approche de placement

Notre approche consiste à placer et router des circuits mixtes. Dans le cas de la partie numérique, le problème consiste à placer un très grand nombre de cellules (pouvant être de l'ordre du millions cellules) avec peu de contraintes de placement. Pour la partie analogique, le problème consiste à placer un petit nombre de modules (jusqu'à une centaine de modules) avec davantage de contraintes (symétrie, appariement, régularité, ...). Dans les deux cas, on cherche toujours à minimiser la longueur des fils d'interconnexions et faire en sorte que le circuit occupe le moins de surface possible.

On rappelle que le placement numérique et le placement analogique sont traités séparément dans notre approche. Il est possible d'utiliser le résultat du placement numérique, effectué par le placeur de **Coriolis** en tant que module dans la phase de placement analogique et mixte. Les travaux de la thèse portent sur le placeur analogique et mixte dont l'approche est présentée dans cette partie.

En étudiant l'état de l'art du placement analogique, nous pensons que certains aspects du placement de la partie analogique nécessitent une attention particulière. C'est en tenant compte des considérations suivantes que nous avons effectué les choix qui ont guidé notre approche :

- **Contrôle du concepteur sur le placeur :** Parmi les approches les plus utilisées de l'état de l'art, le recuit simulé génère des placements respectant des contraintes de placement à partir d'une représentation topologique. Le résultat d'une telle approche nécessite souvent des ajustements ce qui peut être plus ou moins difficile selon la représentation utilisée. L'ajustement des fonction de coûts utilisées par ces approches est généralement réalisé suite à de nombreux essais et ne peut être l'unique manœuvre de jeu du concepteur. Dans notre méthodologie de placement analogique, le placement relatif du circuit est fourni par le concepteur et cette topologie n'est pas perturbée durant la phase de placement. Le concepteur doit utiliser une représentation en *slicing tree* pour définir sa topologie. Définir le *slicing tree* revient à laisser le choix aux concepteurs du placement relatif du circuit.
- **Gestion des contraintes :** On observe dans l'état de l'art que la gestion des contraintes est relativement automatisée. Pour les raisons similaires au cas précédent, il n'est pas évident d'ajuster les fonctions de coûts des méthodes de recuit simulé pour contrôler la gestion des contraintes. Dans notre approche, le concepteur indique avec le *slicing tree* fourni, les contraintes qui y sont appliquées, en particulier les contraintes de symétrie et de régularité. Cela implique que le concepteur contrôle le placement relatif du circuit et les contraintes à respecter (proximité, régularité, contraintes de bordures, placement des modules sensibles aux fortes températures).
- **Interventions du concepteur :** Un placeur trop automatisé laisse peu de contrôle au concepteur sur le résultat du placement. Un facteur qui nous paraît particulièrement important est que l'expérience d'un concepteur peut énormément influencer les choix de placement dans la gestion des compensations des effets parasites. En suivant notre approche, le concepteur choisit la topologie du circuit et les contraintes appliquées, la variation du facteur de forme pour chacun des modules et choisit surtout le placement final parmi plusieurs placements proposés. Certains placements moins compacts pourraient être des placements plus satisfaisants aux yeux

des concepteurs dans certaines situations. Les placements identifiés par notre outil de placement sont disposés sur un graphe où chaque placement est représenté par un point défini par la largeur et hauteur du placement global.

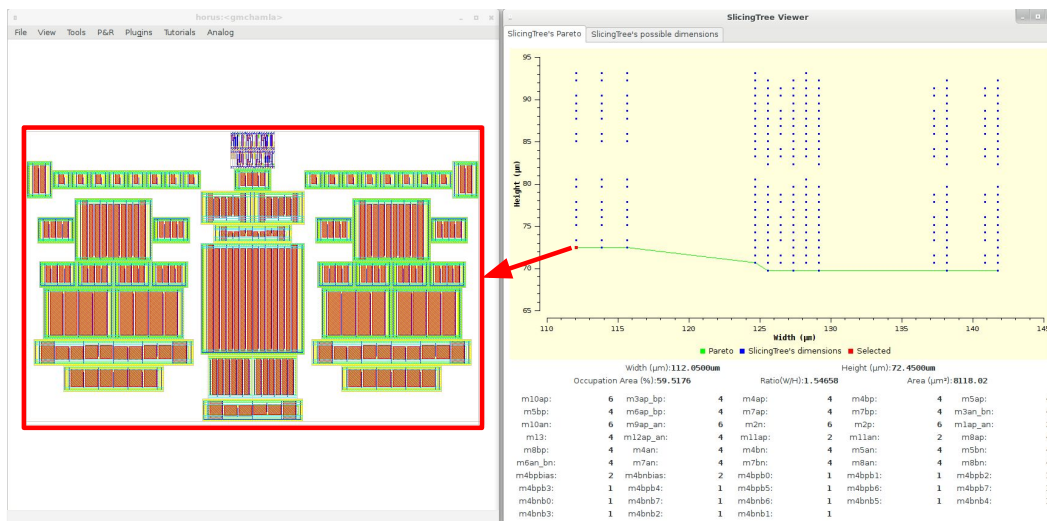
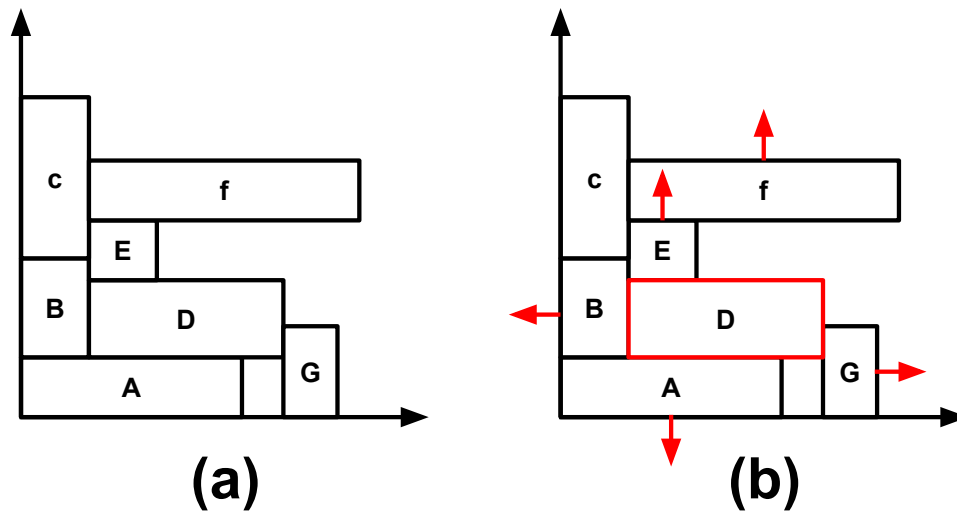


FIGURE 3.17 – Notre approche permet la génération rapide de plusieurs placements

- Temps d'exécution faible :** L'espace des solutions de placement est grand et peut entraîner des longs temps d'exécution pour certains outils de l'état de l'art avec des recherches de solutions pouvant durer plusieurs heures voire plusieurs semaines. Malgré ces temps d'exécution longs, il est possible que le résultat unique obtenu puisse ne pas correspondre aux attentes du concepteur. Pour notre approche, on estime que le temps d'exécution se doit d'être relativement court afin que le concepteur puisse réajuster la topologie de son circuit suite à plusieurs exécutions. Nos temps d'exécution du placement sont de l'ordre de quelques secondes pour des circuits comprenant environ une trentaine de modules permettant aux concepteurs d'effectuer divers essais rapidement.
- Considérations pour la phase de routage :** La majeure partie de l'état de l'art étudie le problème du placement de manière découplée avec la phase de routage. Un circuit trop congestionné ne laissera pas assez de place pour placer les fils de routage. Contrairement aux circuits numériques, certains fils de routage ne peuvent être tracés au-dessus de certaines parties analogiques. Créer des espaces dédiés aux fils de routage en déplaçant les modules d'un circuit très compact peut être un travail extrêmement complexe qui dénaturerait le résultat du placement initial. La figure 3.18 présente un exemple de placement avec une représentation en *HB*-tree* (voir partie 3.3.1) dans lequel il est difficile d'ajouter de l'espace vide autour du module "D" pour des fils de routage par exemple. Pour notre approche, on utilise les caractéristiques de la représentation en *slicing tree* et on utilise ses découpes du plan de masse comme canaux de routage. Ces espaces peuvent être agrandis facilement et ajustés pour le nombre de fils y passant.

Notre méthodologie de placement consiste à effectuer un placement automatisé tout en impliquant des prises de décisions des concepteurs. L'objectif est de permettre à chaque concepteur de pouvoir appliquer ses propres contraintes sans engendrer une automatisation trop avancée. Cela entraînera des résultats moins optimisés en termes de sur-



↑ → ↓ ← : requêtes de déplacement

FIGURE 3.18 – Exemple d'écartement de placement autour du module D avec une représentation en *sequence pair*

face mais en contre partie, le placement sera plus prévisible et plus simplement ajustable. Notre approche repose sur les caractéristiques suivantes :

- **La topologie en *slicing tree*** : Le concepteur définit la position relative des modules à partir d'un *Slicing tree*. Un *Slicing tree* donné implique un placement unique et permet au concepteur de gérer "manuellement" les relations de placement entre certains modules (exemple : placement hiérarchique).
- **Expression des contraintes de placement** : Le concepteur introduit un ensemble de contraintes de placement telles que des contraintes de symétrie ou d'alignement. Le concepteur choisit également les différentes tailles possibles sur chacun de ses modules en déterminant les variations du nombre de doigts possibles pour les transistors des modules.
- **Algorithme de placement** : L'algorithme de placement calcule l'ensemble des placements possibles respectant les contraintes du concepteur. Le concepteur peut alors choisir parmi ces résultats en fonction du facteur de forme et du résultat de placement qui lui convient le mieux.

3.4.2 Définition de la représentation en *slicing tree*

Lors de la conception d'un système sur puce (SoC), les espaces dédiés aux circuits numériques et analogiques sont distincts afin que chacun des circuits puisse être conçu de manière indépendante en termes de surface de circuit. Les circuits numériques sont connus pour leur structure régulière en bandes dans lesquelles les cellules standards sont placées et routées à partir de la *netlist* du circuit en portes. C'est en suivant un placement en bandes que de nombreux circuits analogiques sont également organisés. Notre approche veut également s'inspirer de ce placement en bandes. Pour les circuits analogiques, ces bandes sont en revanche de tailles différentes et dépendent de la hauteur des

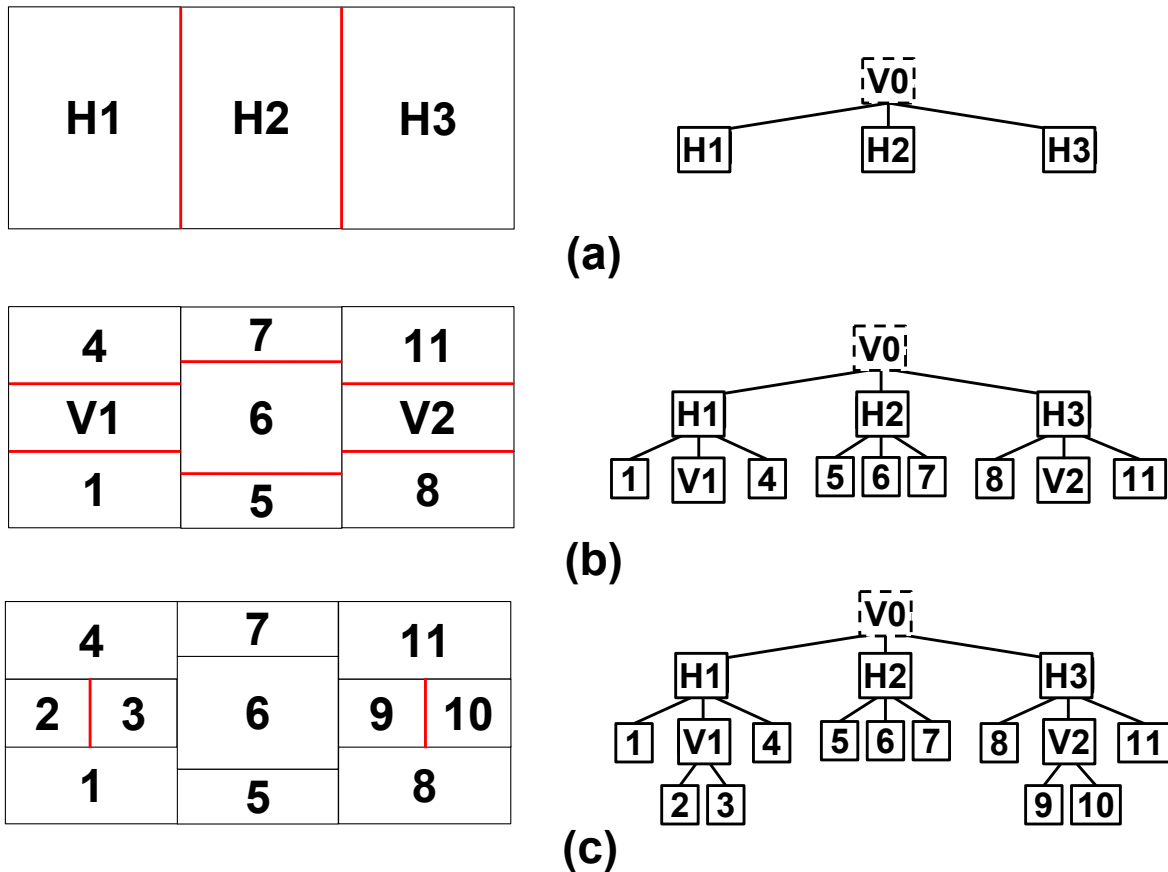


FIGURE 3.19 – Evolution des découpes (indiquées en rouge) d'un slicing tree

modules qui les constituent. Ces bandes ne s'étendent pas d'une extrémité à une autre du circuit comme en numérique mais se limitent à la découpe du circuit, avec une structure en tranches. Placer les modules de cette façon permet d'obtenir des circuits réguliers.

Avec l'objectif de représenter l'organisation en bandes, nous avons opté pour le choix de la représentation en *slicing tree*. Un *slicing tree* permet de définir un espace rectangulaire ayant subi **alternativement** des découpes verticales et horizontales. L'organisation des découpes est décrite sous forme d'un graphe exprimant l'ordre des découpes et dont les feuilles représentent des modules analogiques ou numériques. La figure 3.19 présente un exemple de construction d'un *slicing tree* pour un placement donné. Dans le *slicing tree*, les nœuds commençant par la lettre "V" impliquent que leurs nœuds fils sont le résultat d'une découpe verticale et respectivement ceux avec la lettre "H" impliquent que leurs nœuds fils sont le résultat d'une découpe horizontale. Les nœuds comprenant un numéro uniquement sont des modules :

- figure 3.19.(a) : La surface totale du circuit représenté par la racine "V0" est découpée deux fois créant ainsi trois espaces "H1", "H2" et "H3". Ces trois espaces sont chacun représentés par un nœud hiérarchique dans le *slicing tree*, ces nœuds hiérarchiques sont les nœuds fils du nœud hiérarchique "V0". Les découpes sont indiquées en rouge sur la figure.
- figure 3.19.(b) : Les espaces de chaque nœud hiérarchique "H1", "H2" et "H3" sont chacun découpés deux fois divisant en trois l'espace de chacun de ces nœuds hiérarchiques. On remarque qu'un nœud hiérarchique peut contenir des nœuds hié-

rarchiques et des modules comme pour les nœuds hiérarchiques "H1" et "H3".

- figure 3.19.(c) : Les nœuds "V1" et "V2" sont découpés une fois divisant leur espace d'occupation en deux. Cet exemple comprend au total 11 modules et cette figure montre le *slicing tree* de ce placement.

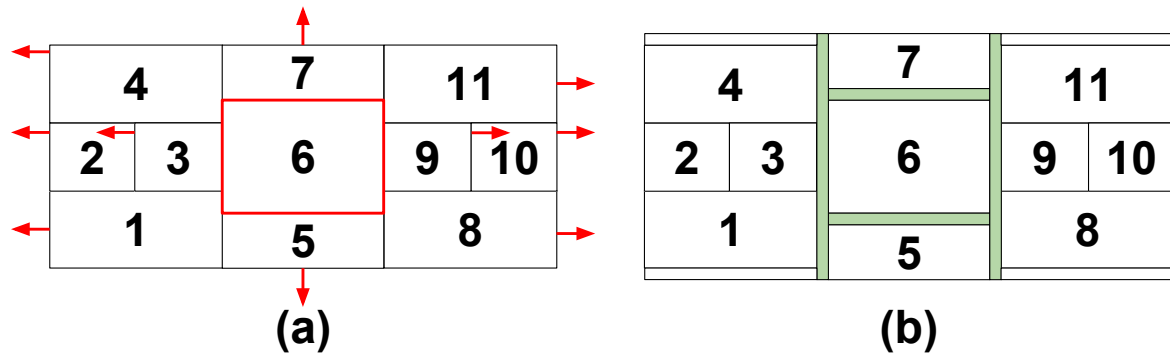


FIGURE 3.20 – Exemple de création d'espaces dédiés aux fils de routage. (a) Le module concerné et le déplacement des nœuds aux alentours du nœud 6 (b) Résultat après écartement et en vert l'espace de routage créé autour du nœud 6

Il revient aux concepteurs la tâche de définir le *slicing tree*. La représentation du *slicing tree* du concepteur ne sera pas altérée par notre outil de placement. Les seules perturbations subies par le placement sont destinés au routage. On rappelle que pour des raisons de diaphonie, les fils de routage ne doivent pas passer au-dessus des zones actives des transistors analogiques, il faut par conséquent les contourner pour les faire passer entre les modules. Dans notre approche de placement et routage, chaque découpe du *slicing tree* définit un canal de routage dont la largeur peut être agrandie. La figure 3.20 illustre un exemple d'écartement des modules agrandissant les canaux de routage autour du module 6. On observe que le placement est perturbé mais que la topologie reste identique au *slicing tree* initial.

3.4.3 Les nœuds du *slicing tree* et leurs contraintes

En définissant eux-même la représentation topologique, les concepteurs auront le contrôle de la topologie et seront libres de juger, avec leur propre expérience, l'importance des contraintes de proximité, régularité ou de bordures en fonction de l'environnement dans lequel se trouve le circuit. Les contraintes de symétries dépendent également de la structure du *slicing tree* mais notre outil de placement garantit le respect des symétries à partir des informations décrites au niveau des nœuds du *slicing tree*. Les éléments qui peuvent constituer un *slicing tree* sont **des devices, les nœuds hiérarchiques, des espaces de routage et des rails traversants.**

Les nœuds *device* et de rails

On définit un *device* comme étant un circuit numérique placé ou un module analogique placé. Pour un module numérique, il peut s'agir d'un circuit numérique contenant une ou plusieurs cellules standards. Pour un module analogique, un *device* représente un ou plusieurs transistors permettant de réaliser une fonction analogique de base (voir

3.2.1). Chacun des *devices* est paramétré par le concepteur qui doit fournir les informations géométriques telles que la taille des transistors ou le style de dessin des masques. Ces modules analogiques sont générés de manière correcte par construction à partir des règles de dessins de la technologie donnée. Pour plus de détails concernant notre bibliothèque de *devices* analogiques, Stéphanie Youssef[38] décrit entièrement le contenu de ces *devices*.

Il est possible de considérer des rails, par exemple pouvant servir lors de la conception de cellule matricielle. On définit comme un rail un fil traversant le circuit d'une extrémité à une autre, la largeur d'un rail est définie par le concepteur. Du point de vue du *slicing tree*, on peut le considérer comme "un *device*" comportant un connecteur unique. La boîte englobante d'un rail est déterminée par l'épaisseur du fil et par la largeur (resp. hauteur) du nœud parent s'il se trouve dans un nœud hiérarchique horizontal (resp. vertical).

Pour l'algorithme de génération du placement, un *device* et un rail d'un *slicing tree* contiennent les attributs suivants (voir figure 3.21) :

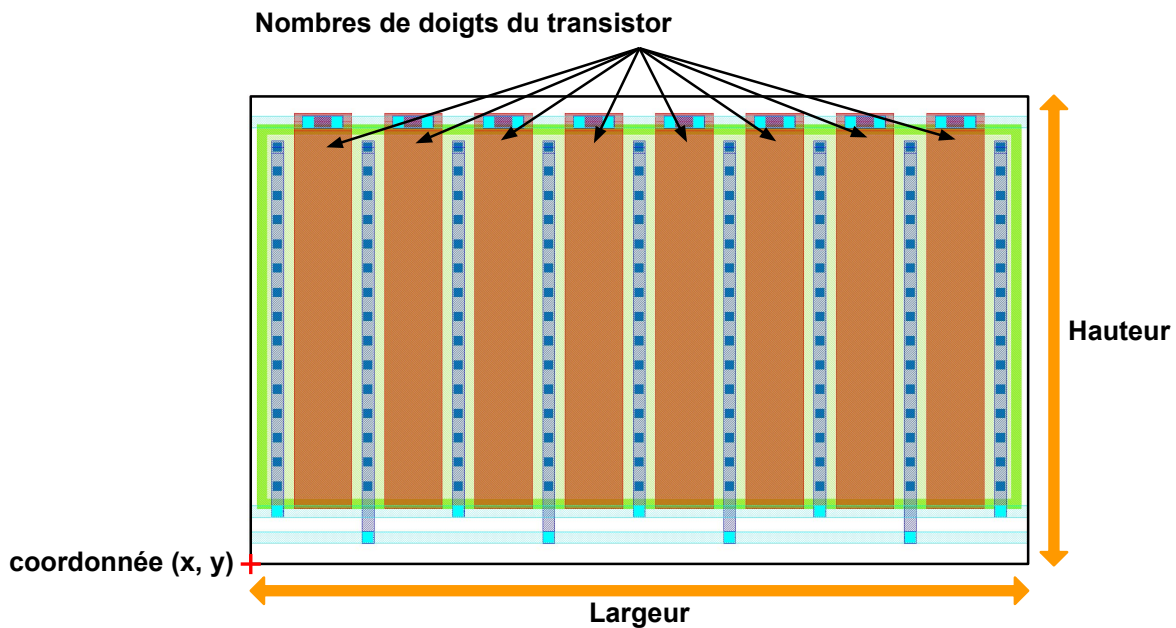


FIGURE 3.21 – Transistor simple à 8 doigts et la représentation de ses attributs.

- **Une position (x, y) :** Elle représente la position du coin inférieur gauche du *device* au sein du plan considéré pour le placement des *devices*.
- **Une hauteur et une largeur :** Elle définit la hauteur et la largeur du rectangle englobant la surface occupée par le *device*/le rail.
- **Une contrainte d'alignement :** Elle positionne le *device* au sein du nœud hiérarchique dont il est le nœud fils. Davantage de détails seront donnés dans la description des nœuds hiérarchiques. Cet attribut n'a pas d'effet sur les rails.
- **Variation du nombre de doigts des transistors :** Les configurations en nombre de doigts de transistors tolérées définies par le concepteur. On souhaite jouer sur la

variation du nombre de doigts des transistors pour utiliser différents facteurs de forme pour les *devices*. Cet attribut ne concerne pas les rails.

Notre approche de placement consiste à organiser les *devices* en bandes et on cherche à faire en sorte que les *device* d'une bande possèdent une hauteur la plus similaire. Afin d'obtenir cette configuration, il est nécessaire de considérer plusieurs facteurs de forme pour chacun des *devices* en faisant varier le nombre de doigts des transistors des blocs analogiques tout en préservant leur fonctionnalité électrique. Néanmoins, nous sommes conscients que cette action implique une variation de la capacité source/drain C_{jSB} pouvant influencer les performances du circuits.

Long Di et al.[22] et Mark Po-Hsun Wu et al.[39] détaillent cette influence de la variation du nombre de doigts d'un transistor sur les performances d'un circuit analogique. Soulignons que notre méthode permet de prendre en compte cette influence en **effectuant plusieurs itérations** de dimensionnement et de génération de dessin des masques. Chaque nouvelle itération permet un raffinement des choix du concepteur sur les paramètres de placement et de routage.

Les nœuds hiérarchiques de découpes verticales et horizontales

Les nœuds hiérarchiques représentent des espaces du circuit résultant de découpes verticales ou horizontales du circuit. Le type du nœud hiérarchique indique les découpes réalisées aux nœuds fils. Si le nœud hiérarchique est du type vertical alors l'espace définissant ce nœud hiérarchique est découpé verticalement. Réciproquement, si le nœud hiérarchique est du type horizontal alors l'espace définissant ce nœud hiérarchique est découpé horizontalement. Ces nœuds permettent de définir la topologie du circuit et de mémoriser les contraintes de symétrie spécifiées par le concepteur.

Pour l'algorithme de génération du placement, un nœud hiérarchique est défini par plusieurs attributs (voir figure 3.22) :

- **Une position (x, y)** : Elle représente la position du coin inférieur gauche de l'espace occupé par le nœud hiérarchique.
- **Une hauteur et une largeur** : Elle définit la hauteur et la largeur de la boîte englobant la surface occupée par le nœud hiérarchique.
- **Une contrainte d'alignement** : Elle positionne le nœud hiérarchique au sein du nœud hiérarchique supérieur dont il est le nœud fils.
- **Ordonnancement des nœuds fils** : Les nœuds fils sont stockés dans un ordre déterminant leur positionnement.
- **Les relations de symétries** : Les nœuds fils symétriques, s'il en existe, sont stockés.
- **Paramètre de validité** : Dans un nœud hiérarchique vertical, ce paramètre définit la différence de hauteurs maximale autorisée entre les nœuds fils pour être considéré comme un placement valide. Réciproquement pour un nœud hiérarchique horizontal, ce paramètre définit la différence de largeurs maximales autorisée entre les nœuds fils pour être considéré comme un placement valide. Plus de détails concernant ce paramètre dans la partie 3.4.4.

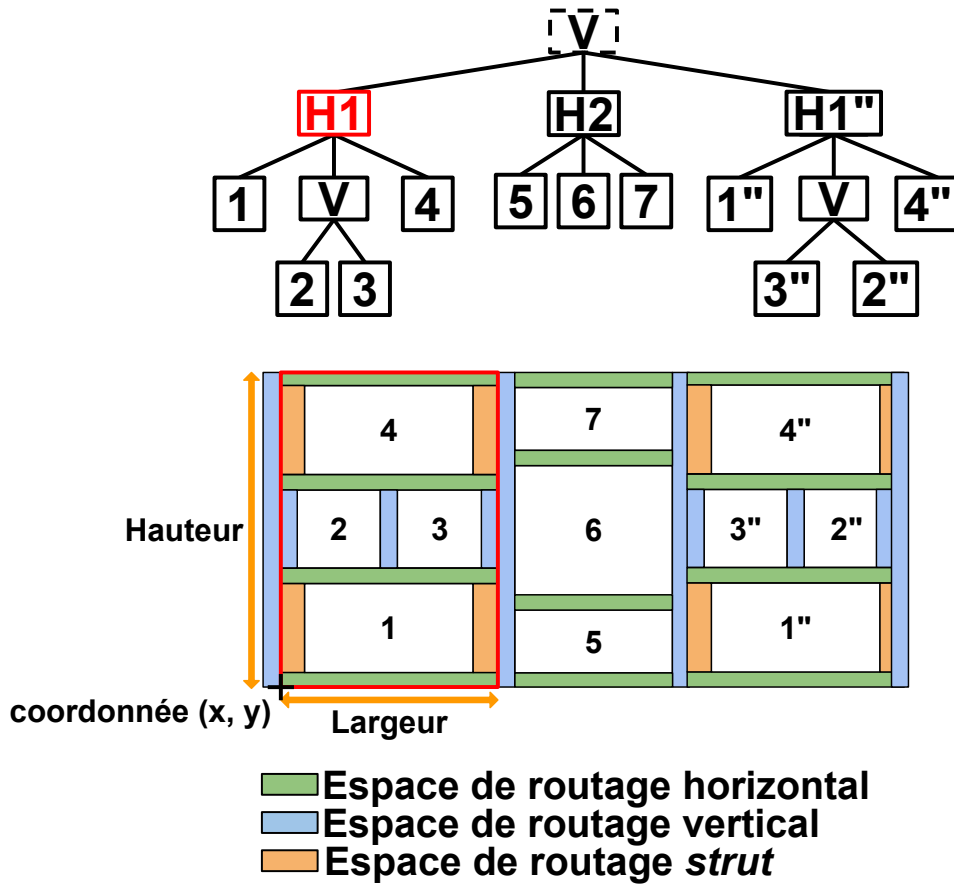


FIGURE 3.22 – Représentation des attributs d'un nœud hiérarchique

Les symétries d'un nœud hiérarchique impliquent que deux nœuds fils possèdent des dimensions (hauteur et largeur) identiques tout au long du traitement de l'algorithme de placement. Cette symétrie est également garantie d'un point de vue graphe, c'est-à-dire que deux nœuds hiérarchiques symétriques possèdent un sous-graphe du *slicing tree* organisé de telle sorte à ce que la symétrie soit possible. Par exemple, sur la figure 3.22, on considère les nœuds "H1" et "H1''" symétriques. Ils possèdent donc la même largeur et hauteur et leur sous-graphe du *slicing tree* est organisé symétriquement (inversion des modules "3''" et "2''").

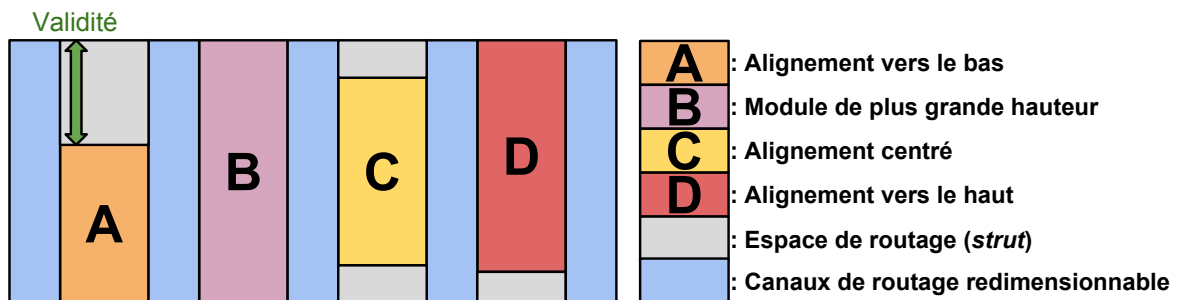


FIGURE 3.23 – Organisation en bande - Exemple de quatre modules (A, B, C et D) d'un nœud hiérarchique vertical

La figure 3.23 présente l'organisation d'un nœud hiérarchique vertical, il s'agit d'un

exemple d'un nœud possédant 4 nœuds fils A, B, C et D. On y observe l'organisation d'une bande dans laquelle les *devices* A, B, C et D sont soumis à des contraintes d'alignement. "A" est aligné vers le bas de la bande, "C" est centré par rapport à la hauteur de la bande et "D" est aligné vers le haut de la bande.

Les nœuds d'espace de routage

On définit comme espace/canal de routage les espaces vides contenus dans la boîte englobant l'ensemble des modules placés du circuit et ces espaces vides sont dédiés au positionnement des fils de routage. Ces espaces/canaux de routage sont issus de deux facteurs :

- **Espace de routage** : La différence de dimensions entre les nœuds fils d'un nœud hiérarchique entraîne la présence d'espaces vides
- **Canaux de routage** : Chaque découpe du circuit est considérée comme un canal de routage rectangulaire redimensionnable dans une seule direction.

Dans le cas d'un canal de routage issu d'une découpe verticale, l'espace de routage peut être redimensionné de manière horizontale (largeur redimensionnable). Réciproquement pour un canal de routage issu d'une découpe horizontale, l'espace de routage peut être redimensionné de manière verticale (hauteur redimensionnable). Les canaux de routage du *slicing tree* sont implicites et ne nécessitent pas d'être paramétrés par le concepteur. Le dimensionnement des canaux de routage est réalisé suite à la phase de routage (plus de détails dans la partie 4.5.6).

Pour être plus précis concernant l'organisation des espaces de routage au sein d'un *slicing tree*, on considère un espace de routage entre chaque nœud fils d'un nœud hiérarchique (horizontal ou vertical). Cela implique que dans un nœud hiérarchique :

$$\text{Nombre d'espace de routage} = \text{Nombre de nœuds fils} + 1 \quad (3.3)$$

L'alternance des canaux de routage et des nœuds fils débute toujours par un espace de routage et se termine toujours par un espace de routage. Durant la totalité de la phase de placement, les canaux de routage redimensionnables sont considérés de largeur nulle.

Pour l'algorithme de génération du placement, un espace de routage est défini par plusieurs attributs (voir figure 3.24) :

- **Une position (x, y)** : Elle représente la position du coin inférieur gauche de la boîte englobante de l'espace de routage au sein du plan considéré.
- **Une hauteur ou une largeur** : Elle définit la largeur pour un canal de routage vertical et la hauteur pour un canal de routage horizontal. Pour les espaces de routage issus de différence de taille de modules, ils sont orientés selon leur dimension : si leur longueur est plus grande que leur hauteur alors ces espaces de routage sont orientés horizontalement et on considère leur hauteur. Réciproquement, si leur hauteur est plus grande que leur longueur, ils sont orientés verticalement et on considère leur largeur.

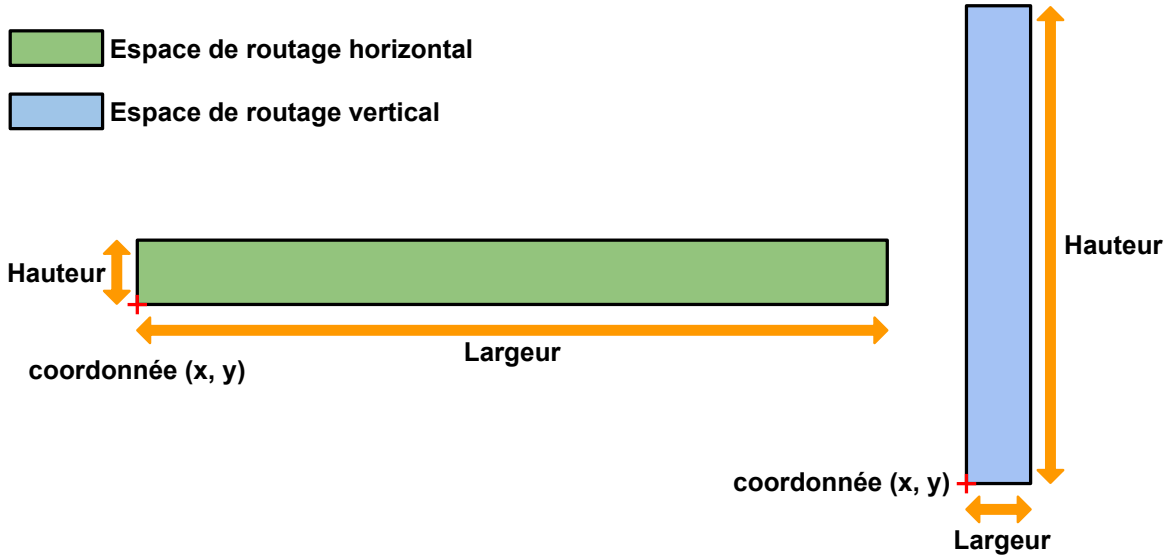


FIGURE 3.24 – Représentation des attributs d'un espace de routage vertical et d'un espace horizontal

3.4.4 L'algorithme de placement

Calcul des positions d'un module à partir d'un *slicing tree*

Soit un *slicing tree* défini par le concepteur, il existe un placement unique lui correspondant et suivant les contraintes d'alignement de chaque nœud du graphe. Afin de comprendre la propagation des facteurs de forme jusqu'à la racine du *slicing tree* permettant de connaître le facteur de forme d'un circuit, nous présentons le calcul du facteur de forme d'un nœud hiérarchique. Soit un *slicing tree*, on considère un nœud hiérarchique comprenant :

- nmh : le nombre de nœuds fils du nœud hiérarchique
- $nœud_i$: le nœud fils à l'index i avec $i \in \{1, \dots, |nmh|\}$
- $(hauteur, largeur)_{nœud_i}$: facteur de forme (hauteur et largeur) du $nœud_i$
- espace de routage $_j$: l'espace de routage à l'index j avec $j \in \{1, \dots, |nmh+1|\}$, l'espace de routage à l'index j se trouve entre les $nœud_{j-1}$ et $nœud_j$

Dans le cas d'un nœud hiérarchique vertical (voir figure 3.25), la hauteur " $hauteur_V$ " et la largeur " $largeur_V$ " du nœud hiérarchique sont calculées de la manière suivante :

$$hauteur_V = \max(hauteur_{nœud_1}, hauteur_{nœud_2}, \dots, hauteur_{nœud_{nmh}}) \quad (3.4a)$$

$$largeur_V = \sum_{i=0}^{nmh} largeur_{nœud_i} + \sum_{j=0}^{nmh+1} largeur_{\text{espace de routage}_j} \quad (3.4b)$$

Dans le cas d'un nœud hiérarchique horizontal (voir figure 3.26), la hauteur " $hauteur_H$ " et la largeur " $largeur_H$ " du nœud hiérarchique sont calculées de la manière suivante :

$$hauteur_H = \sum_{i=0}^{nmh} hauteur_{nœud_i} + \sum_{j=0}^{nmh+1} hauteur_{\text{espace de routage}_j} \quad (3.5a)$$

$$largeur_H = \max(largeur_{nœud_1}, largeur_{nœud_2}, \dots, largeur_{nœud_{nmh}}) \quad (3.5b)$$

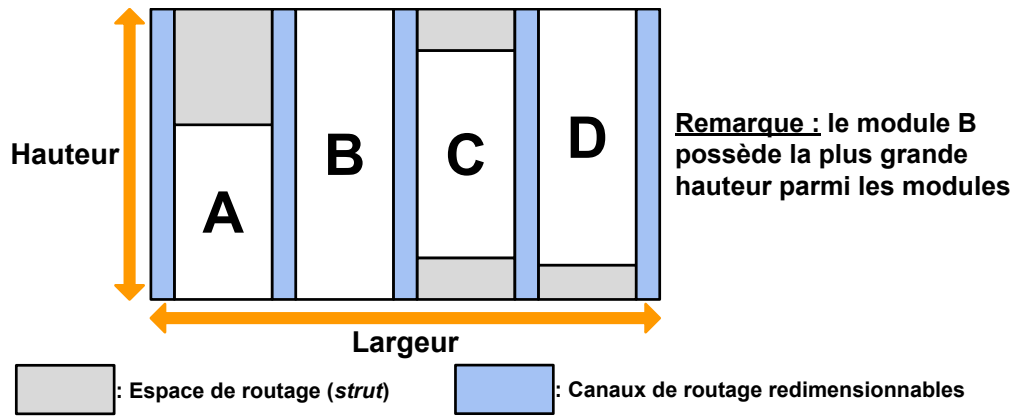


FIGURE 3.25 – Hauteur et largeur d'un nœud hiérarchique vertical

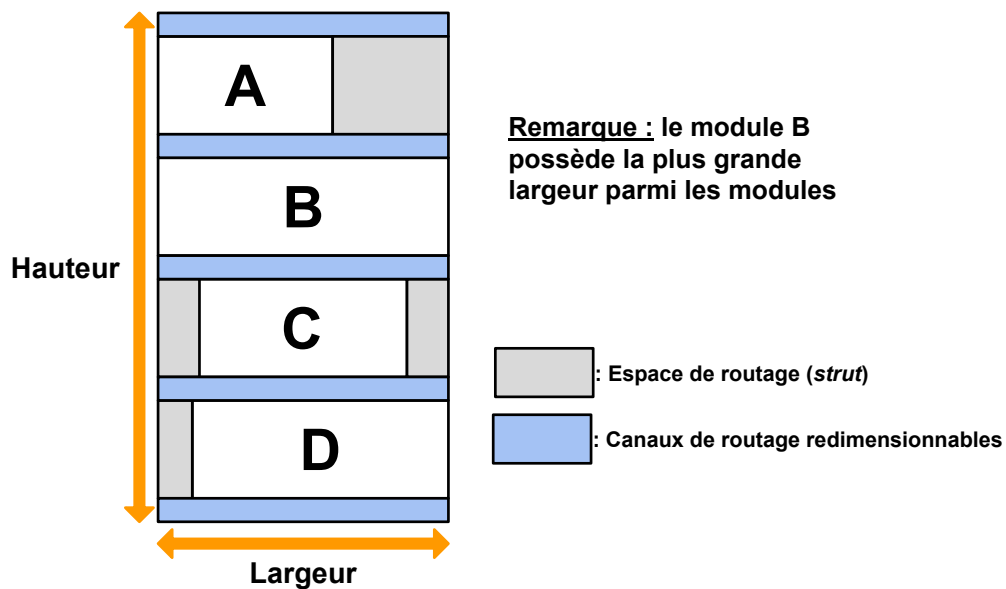


FIGURE 3.26 – Hauteur et largeur d'un nœud hiérarchique horizontal

Le calcul du facteur de forme de chacun des nœuds hiérarchiques est réalisé en partant des feuilles du *slicing tree* et en se propageant vers les nœuds hiérarchiques supérieurs (*bottom-up*). Cela implique que pour calculer le facteur de forme d'un nœud hiérarchique, il est nécessaire que le calcul du facteur de forme de tous ses nœuds fils ait été réalisé auparavant.

Considérations de plusieurs facteurs de forme

Nous jugeons pertinent de proposer plusieurs solutions de placement au concepteur afin de lui permettre d'explorer plusieurs solutions de placement. Par conséquent, le concepteur peut indiquer différents facteurs de forme pour chacun des modules du *slicing tree*. La génération de nos modules analogiques provenant de la librairie de Stéphanie Youssef[2] est contrôlée à partir d'un ensemble de paramètres. En particulier, il est possible de contrôler le nombre de doigts des transistors d'un module analogique. En jouant sur le nombre de doigts des transistors, il est possible de jouer sur le facteur de

forme des modules, le concepteur doit faire le choix des variations du nombre de doigts autorisés.

En considérant plusieurs facteurs de forme pour des modules du *slicing tree*, soit $nm \in \mathbb{N}$ le nombre total de modules, nf_i le nombre de facteurs de forme d'un module $i \in \{1, \dots, nm\}$, le nombre total de facteurs de forme possible sans contraintes considérées est de :

$$\text{Nombre de facteurs de forme total} = nf_1 \times nf_2 \times \dots \times nf_{nm} \quad (3.6)$$

Le nombre de combinaisons à considérer augmente de manière rapide (factorielle) mais certaines considérations réduisant le nombre de possibilités sont à prendre en compte :

- **Symétries des nœuds** : Deux modules symétriques du point de vue du *slicing tree* correspondent à deux nœuds fils d'un nœud hiérarchique ayant les mêmes facteurs de forme paramétrés par le concepteur. Notre outil de placement garantit que ces deux modules prennent toujours les mêmes dimensions, réduisant ainsi le nombre de facteurs de forme à considérer.
- **Paramètre de validité** : Comme mentionné dans les parties précédentes 3.4.3 et 3.4.2, on vise à organiser les *devices* en bandes. On considère que des hauteurs (resp. largeurs) sont similaires lorsque la différence entre la plus petite et la plus grande hauteur (resp. largeur) est inférieure au paramètre de validité. **C'est le rôle des concepteurs de définir ce paramètre de validité afin qu'ils puissent contrôler ce qu'ils considèrent comme une hauteur (resp. largeur) similaire dans un nœud hiérarchique vertical (resp. horizontal).** Ce paramètre éliminera donc des solutions de placements dont les tailles de modules sont trop différentes.
- **Parcours en *bottom-up* du *slicing tree*** : L'algorithme permettant de déterminer les placements respectant le paramètre de validité est basé sur un parcours *bottom-up*. En partant des feuilles du *slicing tree* représentant des modules, les facteurs de forme des modules sont propagés vers les nœuds hiérarchiques supérieurs jusqu'à la racine. Les facteurs de forme retenus au niveau de la racine représentent les facteurs de forme du circuit complet. À chaque niveau hiérarchique, des facteurs de forme peuvent être invalidés à chaque niveau hiérarchique réduisant ainsi le nombre total de facteurs de forme.
- **Nombre de facteurs de forme des modules** : Le nombre de facteurs de forme total est fortement lié aux facteurs de forme tolérés pour chacun des modules analogiques. Ces facteurs de forme considérés pour chacun des modules sont paramétrés par les concepteurs. Ils peuvent donc considérer un nombre raisonnable de facteurs de forme pour chacun des modules, on précise qu'il n'est pas obligatoire que tous les modules possèdent plusieurs facteurs de forme. Il est possible de limiter le facteur de forme à une seule possibilité pour un module. Limiter le nombre de facteurs de forme pour les modules du circuit revient à limiter le nombre de facteurs de forme total.

Tous les facteurs de forme sont propagés avec une approche *bottom-up*. De la même façon que pour trouver le facteur de forme d'un placement, tous les facteurs de forme possibles sont remontés hiérarchiquement au sein du *slicing tree*. Soit un *slicing tree*, on considère un nœud hiérarchique comprenant :

- nmh : le nombre de nœuds fils du nœud hiérarchique

- $nœud_i$: le nœud fils à l'index i avec $i \in \{1, \dots, |nmh|\}$
- nf_i : le nombre de facteurs de forme du $nœud_i$
- $(\text{hauteur, largeur})_{nœud_i}[k]$: facteur de forme (hauteur et largeur) numéro $k \in \{1, \dots, |nf_i|\}$ du $nœud_i$

Chaque nœud hiérarchique mémorise donc un ensemble de facteurs de forme considéré comme valide selon les critères mentionnés dans les parties précédentes :

$$\text{Facteur de forme}_{nœud_i} = \begin{pmatrix} (\text{hauteur, largeur})_{nœud_i}[0] \\ (\text{hauteur, largeur})_{nœud_i}[1] \\ \vdots \\ (\text{hauteur, largeur})_{nœud_i}[nf_i] \end{pmatrix} \quad (3.7)$$

À chaque niveau hiérarchique, toutes les combinaisons des facteurs de forme des nœuds fils sont évaluées. Tous les nœuds fils du nœud hiérarchique doivent être évalués avant l'évaluation des facteurs de forme possibles du nœud hiérarchique. Si deux nœuds fils sont symétriques, les facteurs de forme de ces deux nœuds fils doivent toujours être les mêmes et par conséquent s'ils avaient nf_i facteurs de forme chacun, alors le placeur évalue nf_i facteurs de forme possibles pour ces deux nœuds symétriques au lieu de $nf_{i_1} \times nf_{i_2}$. Une fois l'ensemble des facteurs de forme des nœuds fils évalués, l'algorithme d'évaluation des facteurs de forme se présente de la manière suivante :

Pour un nœud hiérarchique :

for $i = 1 : (nf_1 \times nf_2 \times \dots \times nf_{nmh})$ **do**

 Calcul du facteur de forme à partir de 3.4 pour un nœud hiérarchique vertical et 3.5 pour un nœud hiérarchique horizontal

if *Le paramètre de validité est-il respecté?* **then**

 | Mémorisation du facteur de forme

end

end

Algorithme: Évaluation des facteurs de forme d'un nœud hiérarchique

On précise que la mémorisation d'un facteur de forme au niveau d'un nœud hiérarchique implique la mémorisation des facteurs de forme des nœuds fils utilisés permettant de reproduire ce facteur de forme. Cela permet d'imposer rapidement les facteurs de forme voulus vers les nœuds des feuilles du *slicing tree*, autrement dit les modules du circuit. Plus de détails sont présentés dans la partie suivante concernant l'implémentation.

3.5 Implémentation du placement

Dans cette section, nous détaillons l'implémentation du placement analogique et mixte au sein de la plate-forme Coriolis dont l'ensemble est majoritairement développé en C++ et scripté en langage Python. L'implémentation de notre approche de placement est représentée par la figure 3.27. Comme mentionné dans la partie 3.4.1, le placement numérique et le placement analogique sont réalisés séparément. Le placement numérique est réalisé à partir du placeur numérique de **Coriolis** et doit être réalisé au préalable afin de pouvoir être inséré dans le *slicing tree*. Il est également possible de considérer des cellules

standards utilisées individuellement dans le *slicing tree*. Les étapes du placement analogique et mixte sont les suivantes :

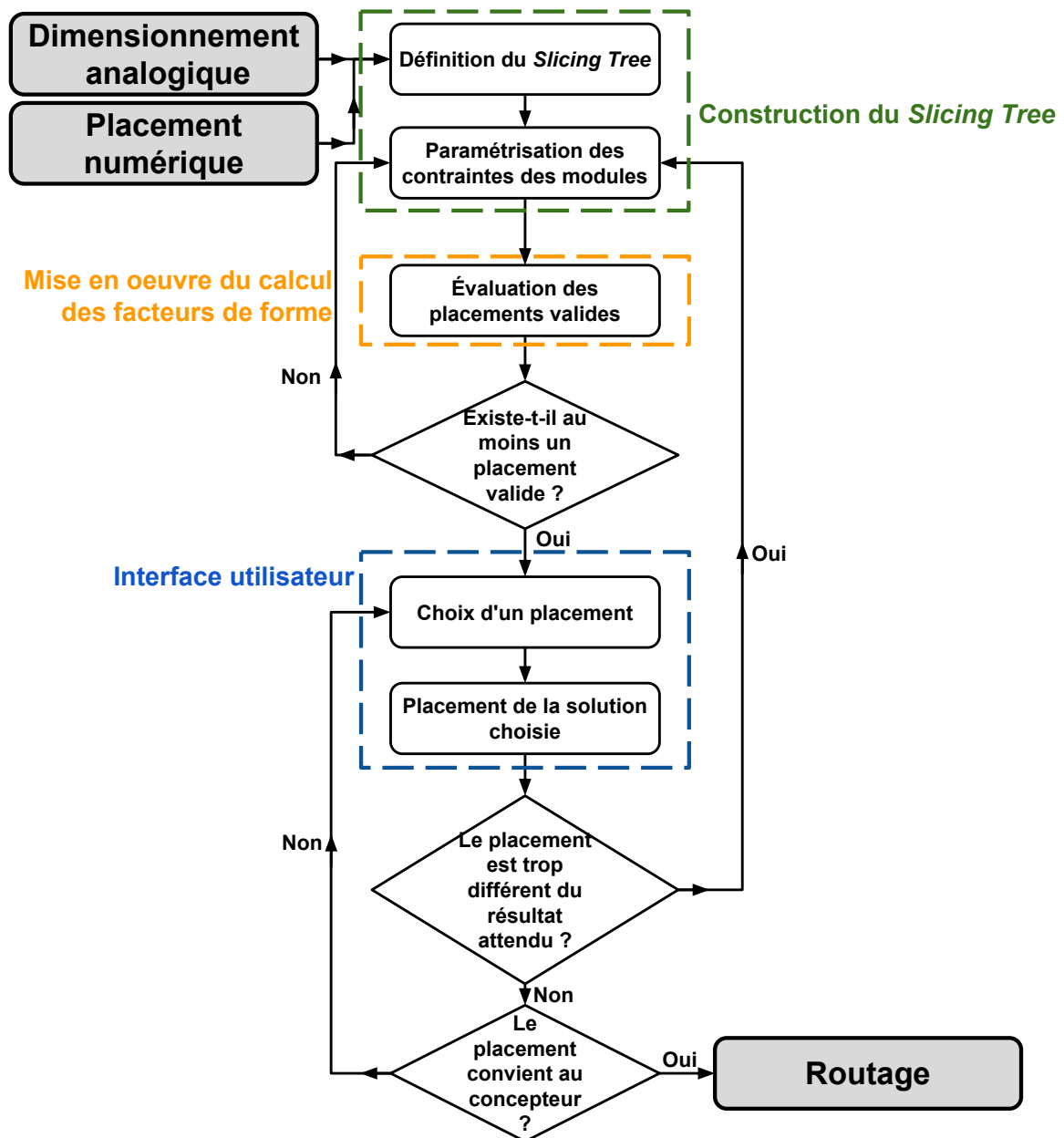


FIGURE 3.27 – Flot du placement analogique et mixte

- **Construction du *slicing tree*** : Cette étape consiste à construire le *slicing tree* à partir de la structure de données dédiée à la gestion du placement analogique et mixte. La construction du *slicing tree* est définie par le concepteur à partir de script Python. Le concepteur impose également la variation du nombre de doigts et les contraintes de symétrie. Les éléments de cette structure de données interagissent avec ceux dédiés à la phase de routage. L'uniformisation des structures de données permet un échange d'informations efficace entre la phase de placement et de routage.
- **Mise en œuvre du calcul des facteurs de forme** : Le calcul des facteurs de forme valide est automatisé et réalisé par notre outil de placement. Les facteurs de forme du circuit complet sont obtenus à partir des contraintes soumises par le concepteur.

On présente dans cette partie les éléments permettant de gérer et mémoriser les facteurs de forme au niveau des nœuds du *slicing tree*.

- **Interface utilisateur** : On présente dans cette partie l'interface utilisateur de **Coriolis** dédiée au placement mais qui sera également utilisée lors de la phase de routage. Les placements considérés comme étant valides y sont présentés ainsi que les informations les concernant telles que les dimensions du placement ou le nombre de doigts choisi pour chacun des modules du circuit s'ils sont analogiques. C'est à partir de cette interface que le concepteur choisit le placement désiré et entraîne la génération du placement de la solution choisie.

Les parties suivantes détaillent l'implémentation de chacune de ces étapes.

3.5.1 Construction du *slicing tree*

Dans le cadre de notre outil de placement analogique et mixte, on utilise un *slicing tree* pour représenter le placement des modules d'un circuit. Le *slicing tree* est défini et paramétré par le concepteur. De ce fait, la structure de données doit être adaptée et pratique en ce qui concerne la description du *slicing tree*. Les éléments permettant de définir le *slicing tree* sont intégrés à **Coriolis** et sont amenés à échanger des informations avec la partie routage.

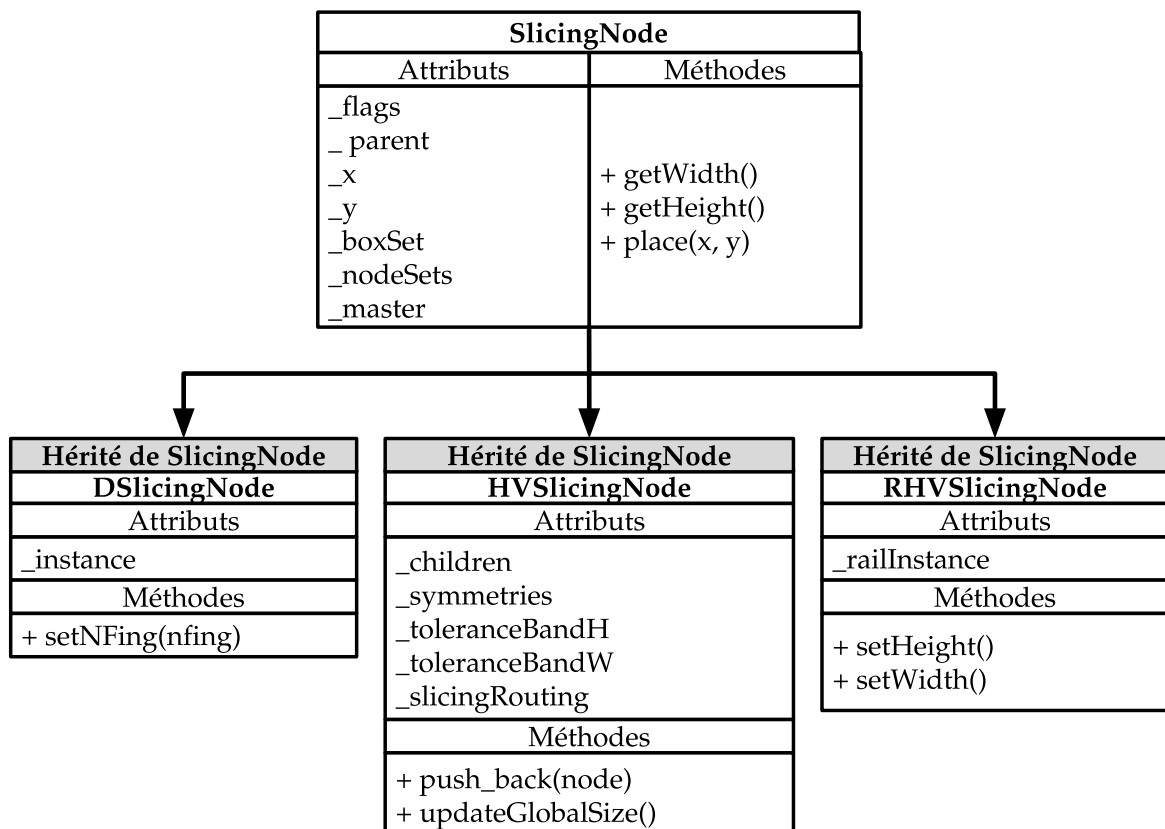


FIGURE 3.28 – Classes utilisées pour la construction du *slicing tree*

La classe *SlicingNode* est la classe abstraite comportant l'ensemble des informations nécessaires pour définir un espace du circuit, qu'il soit un module analogique ou numérique ou un espace/canaux de routage. L'occupation de la surface pour un placement

donné est définie par les modules (*DSlicingNode*) qui sont organisés dans le *slicing tree* par les nœuds hiérarchiques (*HVSlicingNode*) et les espaces/canaux de routage (*RHVSlicingNode*). Les rails traversant le circuit sont représentés à partir de la classe *RHVSlicingNode* également. Nous présentons quelques unes des méthodes et attributs les plus pertinents pour des raisons de clarté. Les méthodes accesseurs et modificateurs ne sont pas listés afin de se limiter au contenu le plus important.

- **SlicingNode** : Cette classe permet de représenter un nœud du *slicing tree* et contient les informations communes à tous les types de nœuds du *slicing tree*. La position du nœud est représentée par les attributs *_x* et *_y*. L'attribut *_parent* fait le lien avec le nœud hiérarchique supérieur et *_master* vers le nœud symétrique s'il existe. Le facteur de forme d'un nœud est défini par un couple de valeurs (hauteur, largeur) et est représenté par l'attribut *_boxSet*. L'attribut *_nodeSets* contient tous les facteurs de forme que le nœud peut prendre. Les méthodes *getWidth()*, *getHeight()*, *place(x, y)* sont définies différemment en fonction du type de nœud traité. L'attribut *_flags* est vecteur de drapeau destiné à des informations variées telles que le type du nœud ou l'alignement du nœud.
- **DSlicingNode** : Cette classe représente un module analogique ou numérique. Si le nœud est un module analogique, l'attribut *_instance* permet d'accéder à la classe réalisant la génération du module dont le nombre de doigts est paramétré par la méthode *setNFing(nfing)*. Un facteur de forme de l'attribut *_boxSet* contient également l'information du nombre de doigts permettant d'obtenir le facteur de forme.
- **HVSlicingNode** : Cette classe représente un nœud hiérarchique du *slicing tree*. L'attribut *_children* est une liste ordonnée contenant les informations des nœuds fils du nœud hiérarchique. Les nœuds symétriques du nœud hiérarchique correspondent aux nœuds fils possédant les mêmes facteurs de forme, l'attribut *_symmetries* contient les index des nœuds fils symétriques. Le paramètre de validité, mentionné dans la partie 3.4.3, vérifiant une différence de hauteur est limité par l'attribut *_toleranceBandH* et celui limitant une différence de largeur par l'attribut *_toleranceBandL*. En plus des nœuds fils, les espaces de routage implicites dus aux découpes de la topologie sont représentés par l'attribut *_slicingRouting*. Le *slicing tree* est construit à partir de la méthode *push_back(node)* en ajoutant en fin de liste un nœud (*node*). *updateGlobalSize()* correspond à la méthode réalisant l'évaluation de tous les facteurs de forme possibles du circuit. Un *HVSlicingNode* est typé soit *horizontal* soit *vertical*.
- **RHVSlicingNode** : Cette classe est utilisée pour représenter un canal de routage issu d'une découpe du *slicing tree* ou un rail traversant le circuit. S'il représente un canal de routage, le concepteur n'a pas à les préciser dans la description de son *slicing tree* car ils sont, on le rappelle, implicites et les méthodes *setHeight()* et *setWidth()* sont utilisées en interne pour ajuster la taille du canal. L'attribut *_railinstance* est considéré uniquement lorsque l'objet représente un rail auquel cas, il représente l'objet de *Coriolis* permettant de représenter un segment. Les méthodes *setHeight()* et *setWidth()* servent dans ce contexte à configurer la largeur du segment. Un *RHVSlicingNode* est typé soit *horizontal* soit *vertical*.

Coriolis a été construit de telle sorte à ce que les parties destinées aux calculs soient écrites en C++ et que le reste, incluant l'interface graphique et les scripts, soit écrit en Python. En particulier, la description du circuit à placer et à router ainsi que le *slicing tree* correspondant sont décrits en Python. Ces scripts Python paramètrent les différentes classes *SlicingNode* présentées ci-dessus décrites en C++. Plus de détails à ce sujet peuvent

être trouvés dans le manuel utilisateur de **Coriolis**[40]

Dans un script *Python*, la description d'un circuit est réalisée par la création d'une classe héritée de la classe *AnalogDesign* qui est une classe contenant toutes les méthodes nécessaires à la description. Le code 3.1 présente un exemple dans lequel est créé un objet "*DesignName*" dont le nom est choisi par le concepteur. La méthode `__init__` est le constructeur par défaut contenant l'initialisation de paramètres internes de la classe et la méthode `build (self, editor)` contient la description du circuit à proprement dit.

Code 3.1 – Création d'un design à placer et router dans *Coriolis*

```

1 class DesignName ( AnalogDesign ):
2     def __init__ ( self ):
3         AnalogDesign.__init__( self )
4         return
5
6     def build ( self, editor ):
7         .
8         .
9         .

```

Dans la méthode `build (self, editor)`, le concepteur est amené à décrire la totalité du circuit, c'est-à-dire décrire la liste des modules analogiques et numériques utilisés, la liste des *nets* internes ou externes et la description du *slicing tree*.

Pour la description des modules du circuit, le concepteur a pour rôle de définir les caractéristiques géométriques de ses modules. L'attribut `self.devicesSpecs`, comme présenté dans le code 3.2, décrit à partir de tuples dont les arguments pour un module analogique sont décrits de la manière suivante :

Code 3.2 – Description des modules analogiques et numérique du circuit

```

1 self.devicesSpecs = [ [ deviceclass , 'name' , layoutStyle
2                       , transistorType , W, L, dummy
3                       , sourceFirst , bulk , bulkConnected ]
4                       .
5                       .
6                       .
7                       , [ cell , name ]
8                       ]

```

- **deviceclass** : Fonction réalisée par le module (voir liste dans la partie 3.2.1).
- **name** : Chaîne de caractère indiquant le nom du module.
- **layoutStyle** : Style du dessin des masques (par exemple : interdigité).
- **transistorType** : Transistor(s) NMOS ou PMOS.
- **W, L** : Dimensions (largeur et longueur) du/des transistor(s).
- **dummy** : Nombre de transistors factices ajoutés de chaque côté du peigne du module.
- **sourceFirst** : Drapeau indiquant si la source est la zone de diffusion gauche du premier transistor à gauche du peigne.

- **bulk** : Drapeau sur quatre bits indiquant la présence d’anneaux de garde pour chaque côté du module (nord, sud, est, ouest).
- **bulkConnected** : Drapeau indiquant si le connecteur *bulk* est connecté à la source sinon on considère le *bulk* comme un connecteur.

On rappelle que la génération de chacun des modules est le résultat de précédent travaux de la thèse de Stéphanie Youssef[2]. Pour un module numérique placé, le tuple doit contenir l’objet décrivant le circuit numérique (*cell*) et le nom en chaîne de caractère indiquant le nom du module (*name*). On rappelle que notre approche consiste à effectuer plusieurs itérations des étapes de dimensionnement et de placement-routage afin de permettre des ajustements/améliorations après chaque itération. Dans la mesure où le dimensionnement est réalisé avec l’outil OCEANE[41], le fichier de sortie contenant les dimensions des transistors peut être chargé à partir de la méthode *readParameters(filepath)* dont l’argument *filepath* correspond au chemin vers le fichier de sortie d’OCEANE. Cela permet un chargement automatique des valeurs de dimensions des modules du circuit. Cette méthode doit être appelée suite à la définition des modules du circuit.

Pour la description des *nets*, le concepteur a pour rôle de définir les noms des *nets* et s’ils sont des *nets* internes au circuit ou s’ils ont une connexion vers l’extérieur. La description détaillée des *nets* sera présentée dans la partie 4.5.7 de ce manuscrit. Le code 3.3 présente la syntaxe de description des *nets*.

Code 3.3 – Description des *nets* du circuit

```

1 self.netTypes = { 'net1': { 'isExternal':True }
2                   .
3                   .
4                   .
5                   , 'net2': { 'isExternal':False }
6                   }

```

Suite à la description des modules et des *nets*, la description des modules et des *nets* du circuit "*DesignName*" est chargé comme présenté dans le code 3.4 avec les méthodes *self.doDevices()* et *self.doNets()*. La description du *slicing tree* est contenue entre les méthodes *self.beginSlicingTree()* et *self.endSlicingTree()*. Pour les paramètres de validité d’une bande mentionnés dans la partie 3.4.3, les méthodes *self.setToleranceBandH(valueH)* et *self.setToleranceBandW(valueW)* permettent de les définir respectivement pour la validité dans un nœud hiérarchique vertical et horizontal. Une fois les facteurs de forme connus pour le circuit complet, il est possible d’automatiser le placement en choisissant les dimensions désirées (*height* et *width*).

Code 3.4 – Initialisation de la description du circuit et du *slicing tree*

```

1 self.beginCell( 'DesignName' )
2 self.doDevices()
3 self.doNets()
4 self.beginSlicingTree()
5 self.setToleranceBandH( valueH )
6 self.setToleranceBandW( valueW )
7     .
8     .
9     .

```

```

10 self.endSlicingTree ()
11 self.updatePlacement( height , weight )
12 self.endCell ()

```

La construction du *slicing tree* est réalisée de manière hiérarchique. La description d'un nœud hiérarchique est comprise entre les méthodes *self.pushHNode(alignment)* pour un nœud hiérarchique horizontal, *self.pushVNode(alignment)* pour un nœud hiérarchique vertical et la méthode *self.popNode()*. Le paramètre *alignment* indique la contrainte d'alignement du nœud dans son nœud hiérarchique supérieur.

Code 3.5 – Nœud hiérarchique horizontal

```

1 self.pushHNode( alignment )
2     .
3     .
4     .
5 self.popNode ()

```

Code 3.6 – Nœud hiérarchique vertical

```

1 self.pushVNode( alignment )
2     .
3     .
4     .
5 self.popNode ()

```

Dans la description d'un nœud hiérarchique, les relations de symétrie sont indiquées en utilisant la méthode *self.addSymmetry(index, indexCopy)* (voir code 3.7. *index* représente l'index du nœud maître dans le nœud hiérarchique et *indexCopy* représente l'index du nœud symétrique dans le nœud hiérarchique. L'ordre de description implique l'ordre de positionnement en tant que nœud fils dont les index sont compris entre 0 et [le nombre de nœud fils-1].

Code 3.7 – Contrainte de symétrie entre deux nœuds fils

```

1 self.addSymmetry( index , indexCopy )

```

Dans la description d'un nœud hiérarchique, l'ajout d'un module est réalisé à partir de la méthode *self.addDevice*. Les arguments d'ajout d'un module du code 3.8 sont les suivants :

Code 3.8 – Module analogique

```

1 self.addDevice( 'name' , alignment
2                 , span=(start , step , count)
3                 , NF = value )

```

- **'name'** : Nom du module précédemment indiqué.
- **alignment** : Contrainte d'alignement du nœud dans son nœud hiérarchique supérieur.
- **span (utilisé si le module est analogique)** : Paramètre de variation du nombre de doigts, *start* correspond au nombre de doigts initial. *step* correspond au pas d'évolution du nombre de doigts et *count* correspond au nombre d'incrémentations du nombre de doigts réalisées avec le pas *step*.
- **NF (utilisé si le module est analogique)** : Nombre de doigts du transistor si le module analogique n'a pas de variation de facteur de forme.

Comme mentionné dans la partie 3.4.3, les rails sont des nœuds similaires aux nœuds représentant un module, c'est-à-dire qu'ils correspondent à des feuilles du *slicing tree*. L'ajout d'un rail traversant est réalisé en utilisant la méthode *self.addHRail*, la syntaxe est illustrée par le code 3.9 dont les arguments sont les suivants :

Code 3.9 – Rail horizontal

```

1 self.addHRail(
2     self.getNet('net1')
3     , 'METALX', value
4     , "cellName"
5     , "instanceName" )

```

Code 3.10 – Rail vertical

```

1 self.addVRail(
2     self.getNet('net2')
3     , 'METALX', value
4     , "cellName"
5     , "instanceName" )

```

- `self.getNet('net1')` : *net* du rail.
- `'METALX'` : Niveau de métal 'X'.
- `value` : hauteur du rail pour un rail horizontal et largeur du rail pour un rail vertical.
- `cellName` et `instanceName` : nom du rail au sein de *Coriolis*.

3.5.2 Mise en œuvre de l'algorithme de placement

L'algorithme permettant de calculer les facteurs de forme pour chaque nœud hiérarchique est implémenté en C++ et repose sur la méthode `updateGlobalSize()` mentionnée précédemment dans la partie 3.5.1. L'algorithme de la méthode `updateGlobalSize()` se présente de la manière suivante avec le code 3.11 :

Code 3.11 – Méthode `updateGlobalSize()`

```

1 void HSlicingNode::updateGlobalSize()
2 {
3     // Calculer les facteurs de forme des noeuds fils en premier lieu
4     vector<SlicingNode*>::iterator it = _children.begin()
5     for (; it != _children.end(); it++) {
6         (*it)->updateGlobalSize();
7     }
8     // Verification que le noeud courant est un noeud symetrique
9     if (this->getMaster() == NULL) {
10        HSetState state = HSetState(this);
11        while( !state.end() ){ state.next(); }
12        _nodeSets = state.getNodeSets();
13    } else {
14        // Copie des facteurs de forme du noeud maitre
15        _nodeSets = _master->getNodeSets();
16    }
17 }

```

1. **Calcul des facteurs de forme des nœuds fils** : La méthode est appliquée à l'ensemble des nœuds fils. Les nœuds feuilles d'un *slicing tree* correspondent à des nœuds représentant des modules dont les facteurs de forme sont choisis par le concepteur. Le calcul des facteurs de forme se propage donc en *bottom-up*.
2. **Vérification si le nœud courant est symétrique** : Le calcul des facteurs de forme s'applique uniquement au nœud hiérarchique n'étant pas le symétrique d'un autre nœud. Un nœud symétrique copie directement les facteurs de forme du nœud maître.
3. **Calcul des facteurs de forme du nœud courant** : La classe *HVSetState* consiste à parcourir les facteurs de forme pour calculer les facteurs de forme du nœud courant. L'objet *HVSetState* vérifie si le paramètre de validité est respecté et construit

le conteneur *_nodeSets* contenant les facteurs de forme que peut prendre le nœud hiérarchique.

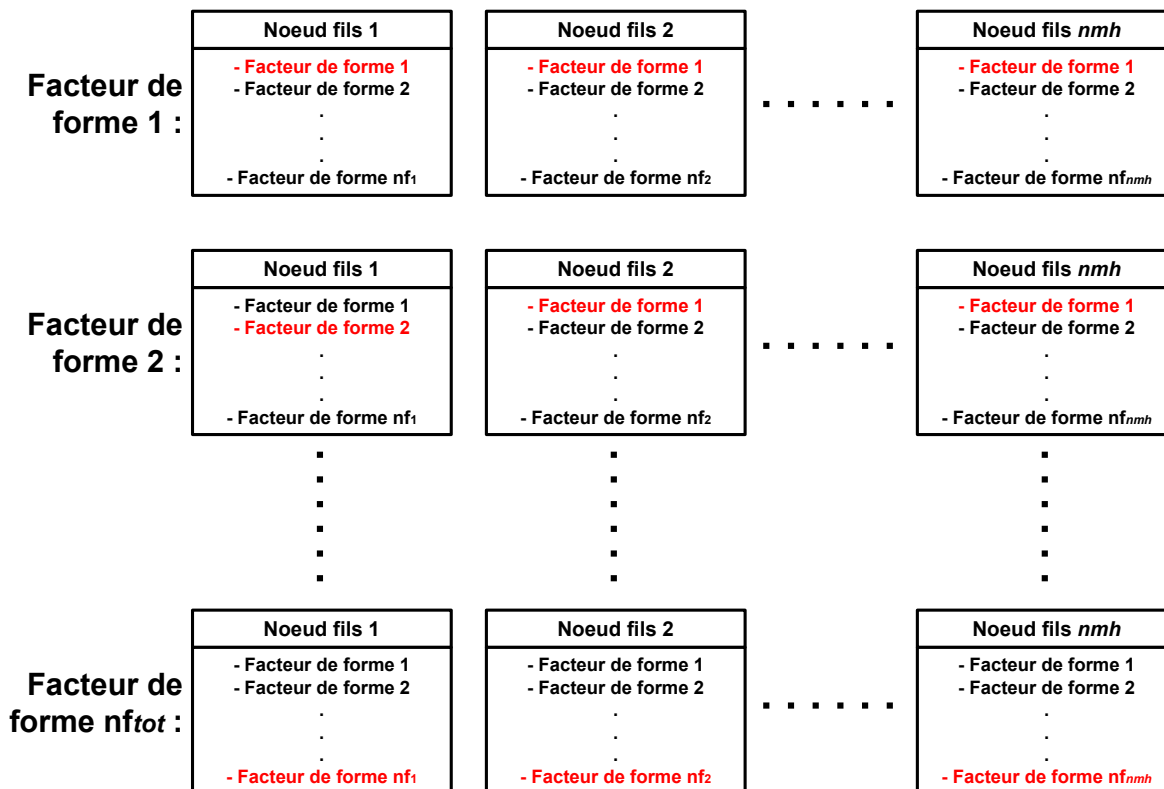


FIGURE 3.29 – Parcours des facteurs de forme possibles

Pour explorer de tous les facteurs de forme valides au niveau d'un nœud hiérarchique, on utilise la classe *HVSetState* dont le fonctionnement est semblable à une machine à état. Cette classe permet de déterminer un facteur de forme, plus précisément le choix du facteur de forme de chacun des nœuds fils parmi chacune de leurs possibilités et de passer à la combinaison suivante.

Soit un nœud hiérarchique et ses nœuds fils dont les facteurs de forme sont calculés, le calcul du facteur de forme du nœud hiérarchique consiste à calculer la hauteur et la largeur du nœud hiérarchique pour toutes les combinaisons de facteurs de forme possibles des nœuds fils. Le parcours des combinaisons est réalisé suivant le modèle de la figure 3.29. Tous les nœuds fils sont initialisés avec leur premier facteur de forme. Pour obtenir la combinaison suivante, on incrémente l'index du facteur de forme du premier nœud fil. Une fois l'ensemble des facteurs de forme du premier nœud fils parcouru, le premier nœud fils retourne à son premier facteur de forme et le second nœud fils incrémente l'index de son facteur de forme.

De manière générale, le premier nœud voit son index de facteur forme incrémenté après chaque combinaison. Avant chaque incrémentation, le facteur de forme du nœud hiérarchique est calculé à partir de la combinaison de facteur de forme des nœuds fils et si il est validé il est ajouté à un conteneur *nodeSets* qui est une liste de *boxSet* comprenant les informations de facteurs de forme que les nœuds fils doivent prendre pour obtenir le facteur de forme courant. Concernant l'incrémentement de l'index du facteur

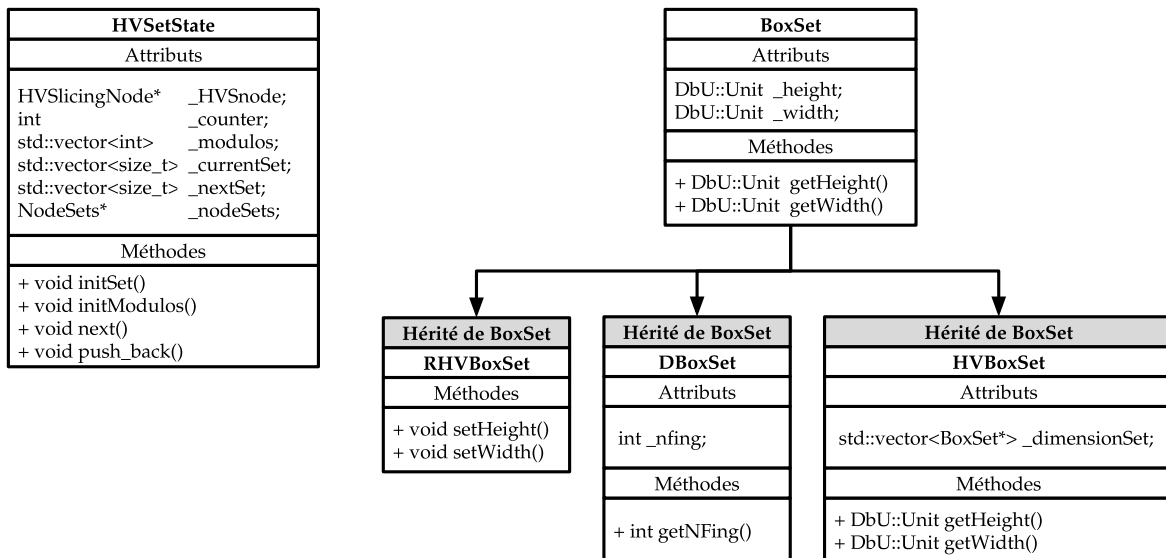


FIGURE 3.30 – Classes utilisées pour évaluer et représenter les facteurs de forme

de forme des autres nœuds fils, une fois le dernier facteur de forme d'un nœud fils n ($n \in nmh$: nombre de nœuds fils dans le nœud hiérarchique) est atteint, cela entraîne le l'incrémement du nœud fils $n+1$. L'algorithme s'arrête après le traitement de la combinaison de facteurs de forme comprenant le dernier facteur de forme de chaque nœud fils.

Les attributs et les méthodes de la classe *HVSetState* sont présentés par la figure 3.30 :

- **_HVSNode** : Nœud hiérarchique traité,
- **_counter** : Compteur du nombre de facteurs de forme traités.
- **_modulos** : Conteneur du nombre de facteurs de forme pour chacun des nœud fils.
- **_currentSet** : Facteur de forme en cours de traitement.
- **_nextSet** : Prochain facteur de forme à traiter.
- **_nodeSets** : Ensemble des facteurs de forme retenus.
- **initSet()** : Initialise *_currentSet* pour traiter le premier facteur de forme.
- **initModulos()** : Initialise l'attribut *_modulos*.
- **push_back()** : Vérification de la bande avec le paramètre de validité du facteur de forme en cours (*_currentSet*) et ajout à *_nodeSets* si le facteur de forme est validé.
- **next()** : Appel de la méthode *push_back()* et passe au prochain facteur de forme.
- **end()** : Vérification si le dernier facteur de forme a été traité.

Sur le code 3.11, on observe l'utilisation de la classe *HVSetState*. Une fois les facteurs déterminés pour tous les nœuds fils, la classe *HVSetState* est créée à partir du nœud hiérarchique courant. Son constructeur fait appel aux méthodes *initSet()* et *initModulos()* permettant d'initialiser l'algorithme. La méthode *next()* vérifie le facteur de forme courant, s'il respecte les contraintes de validité de bande et prépare le prochain facteur de forme à vérifier. Cette méthode est répétée au sein de la boucle *while* dont la condition impose que le dernier facteur ait été traité.

La mémorisation d'un facteur de forme au niveau d'un nœud hiérarchique correspond à une mémorisation de lien vers les facteurs de forme des nœuds fils permettant

d'obtenir les dimensions du facteur de forme. Un facteur de forme est représenté par un objet *BoxSet* (voir figure 3.30) dont hérite les classes *RHVBoxset*, *DBoxSet* et *HVBoxSet* :

- **classe *BoxSet*** : La classe *BoxSet* contient deux attributs représentant la hauteur (*_height*) et la largeur *_width* d'un nœud du *slicing tree* et deux méthodes permettant d'y accéder. Le type *DbU* : *Unit* correspond à l'unité symbolique utilisée dans la structure de données **Coriolis**.
- **classe *RHVBoxset*** : Cette classe permet de représenter le facteur de forme d'un espace de routage. Elle possède deux méthodes permettant de dimensionner le canal de routage à partir de la méthode *setHeight()* si le nœud est un canal de routage horizontal et de la méthode *setWidth()* si le nœud est un canal de routage vertical.
- **classe *DBoxSet*** : Cette classe permet de représenter le facteur de forme d'un nœud représentant un module. Le facteur de forme d'un module est déterminé par le nombre de doigts de ses transistors indiqué par l'attribut *_nfing*.
- **classe *HVBoxSet*** : Cette classe permet de représenter le facteur de forme d'un nœud hiérarchique. Elle possède un attribut supplémentaire *_dimensionSet* permettant de mémoriser une combinaison de *BoxSet* de ses nœuds fils. Les méthodes *getHeight()* and *getWidth()* permettant d'obtenir la hauteur et la largeur du nœud sont obtenus à partir du vecteur de pointeurs vers les *BoxSets* des nœuds fils (voir figure 3.31).

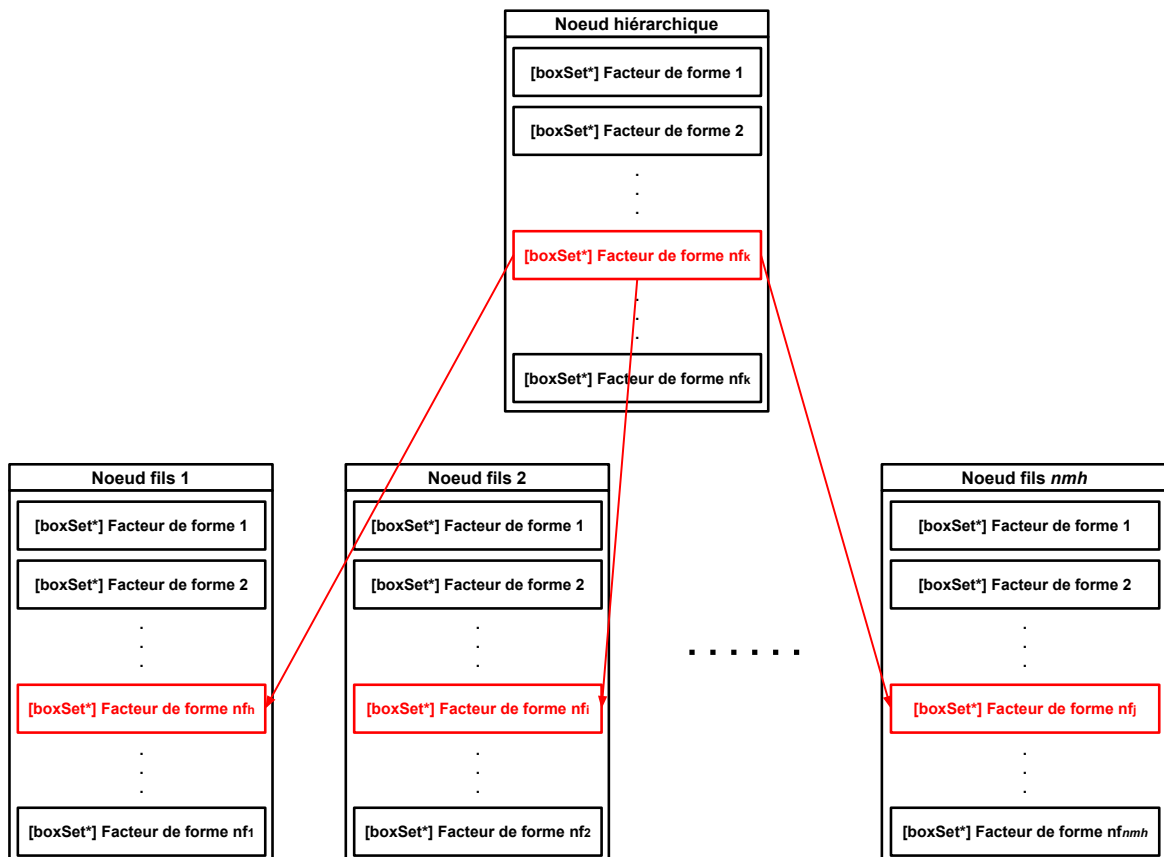


FIGURE 3.31 – Facteur de forme d'un nœud hiérarchique et celui de ses nœuds fils

Cette architecture logicielle permet de simplifier la méthode de placement du *slicing tree*. Un *BoxSet* au niveau d'un nœud hiérarchique correspond à un graphe indiquant les facteurs de forme des niveaux hiérarchiques inférieurs. Connaissant le facteur de forme

de tous les modules du circuit et de leurs alignements, il est possible de déterminer le placement correspondant à partir de la structure du *slicing tree*. Un placement est réalisé en parcourant le graphe d'un facteur de forme *BoxSet*. Les dimensions des espaces de routage, créés à cause des différentes de tailles de modules, sont mises à jour au sein du *slicing tree* lors de la phase de placement.

3.5.3 Interface utilisateur

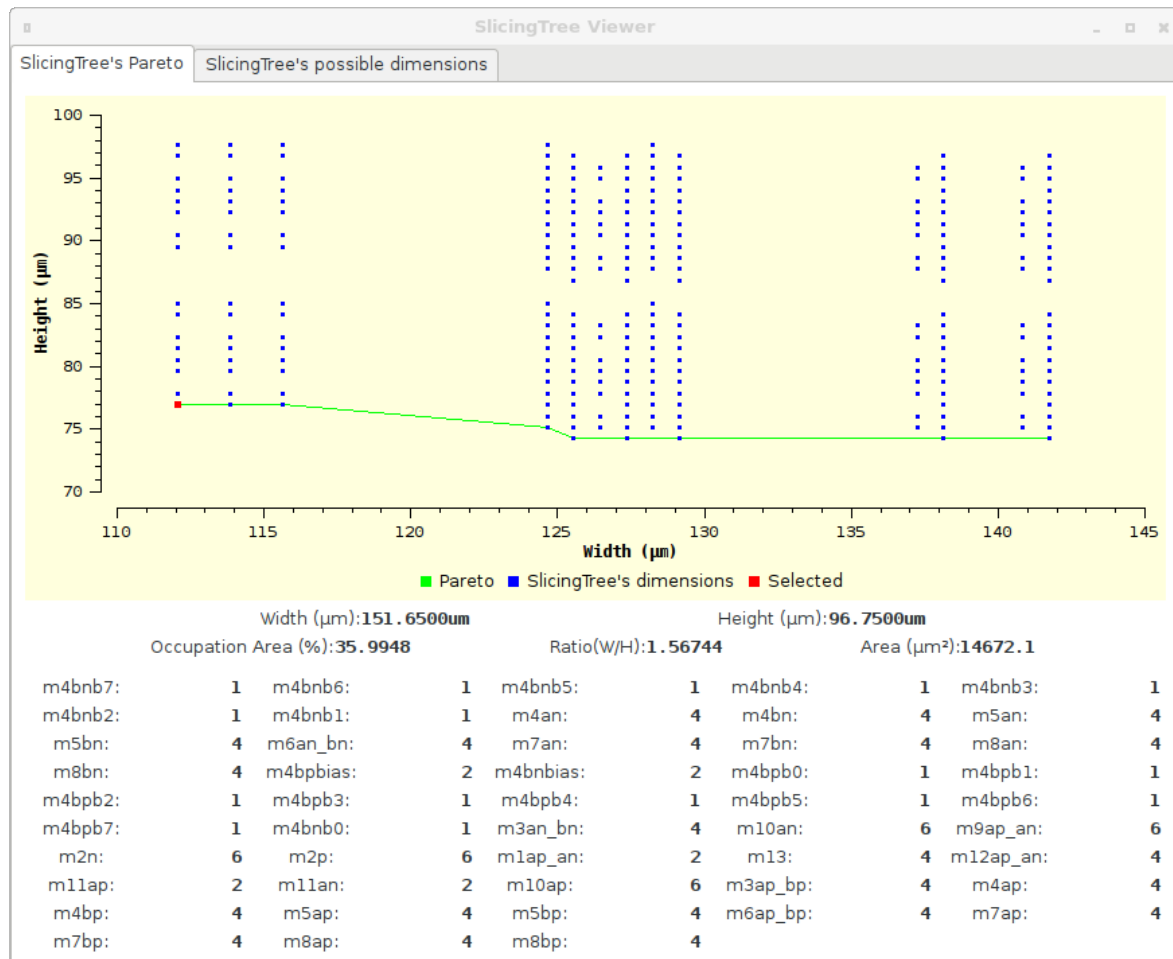


FIGURE 3.32 – Interface utilisateur - Graphe de points des facteurs de forme

Suite à l'évaluation des facteurs de forme, le placeur est en mesure de proposer l'ensemble des facteurs de forme considérés comme étant valide, c'est-à-dire respectant les contraintes de symétrie et des paramètres de validité de bande (voir partie 3.4.4). Le placeur présente tous les facteurs de forme possibles à travers une interface graphique. Ces facteurs de forme sont organisés sous forme d'un graphe de points sur lequel chaque point représente un placement possible défini par ses coordonnées représentant la largeur (coordonnée en abscisse) et la hauteur (coordonnée en ordonnée) du placement. Le concepteur choisit un placement en cliquant dessus et l'outil génère automatiquement le placement induit et l'affiche dans la fenêtre du visualisateur en quelques secondes. Le concepteur peut alors observer si le placement induit lui convient ou choisir un autre placement en cliquant sur un autre point si le placement ne répond pas à ses attentes. L'ensemble des placements est également présenté sous forme d'un tableau contenant

chaque placement possible.

La figure 3.32 présente la fenêtre affichant les différents facteurs de forme. Sur cet exemple, le point le plus en bas et à gauche en rouge est le placement sélectionné. La partie inférieure, juste sous le graphe de points, indique les informations concernant le placement choisi et indique pour chacun des modules, le nombre de doigts utilisé. Par exemple, on peut voir que le module *m10ap* a des transistors à 6 doigts.

La figure 3.33 présente la fenêtre présentant le placement induit par le point choisi. Ce visualisateur permet d'inspecter les différents aspects du circuit et fait partie intégrante de l'outil **Coriolis** (plus de détails à ce sujet peuvent être trouvés dans le manuel d'utilisateur [40]).

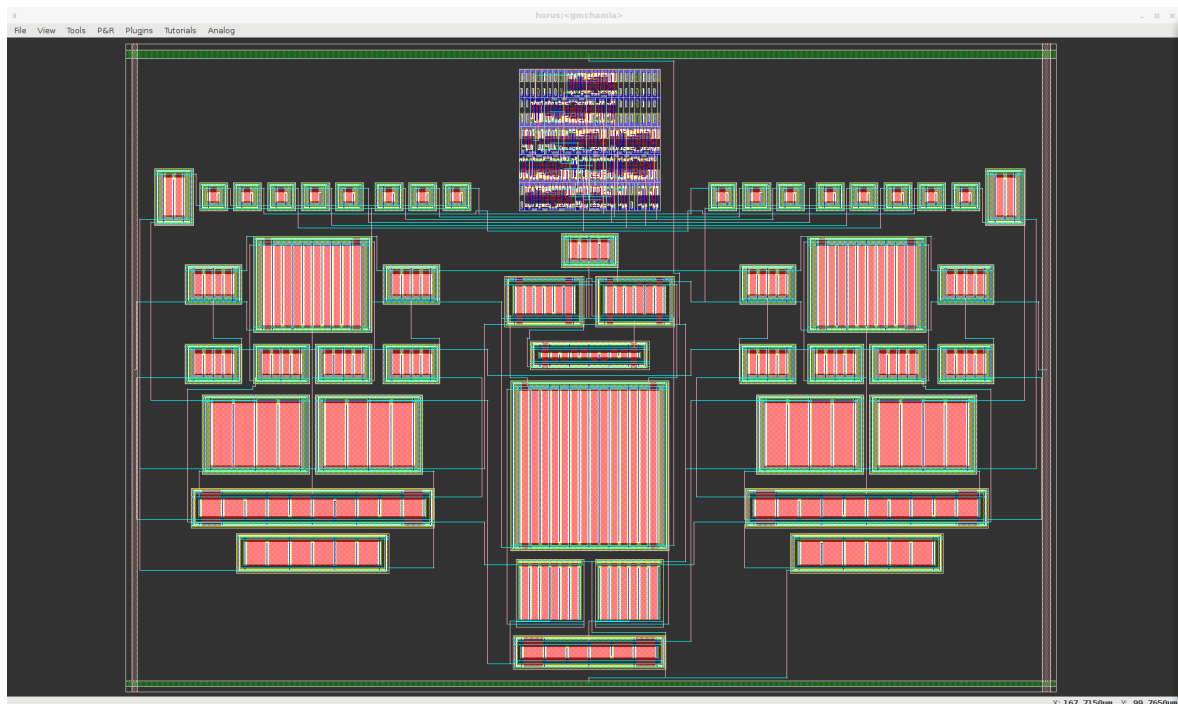
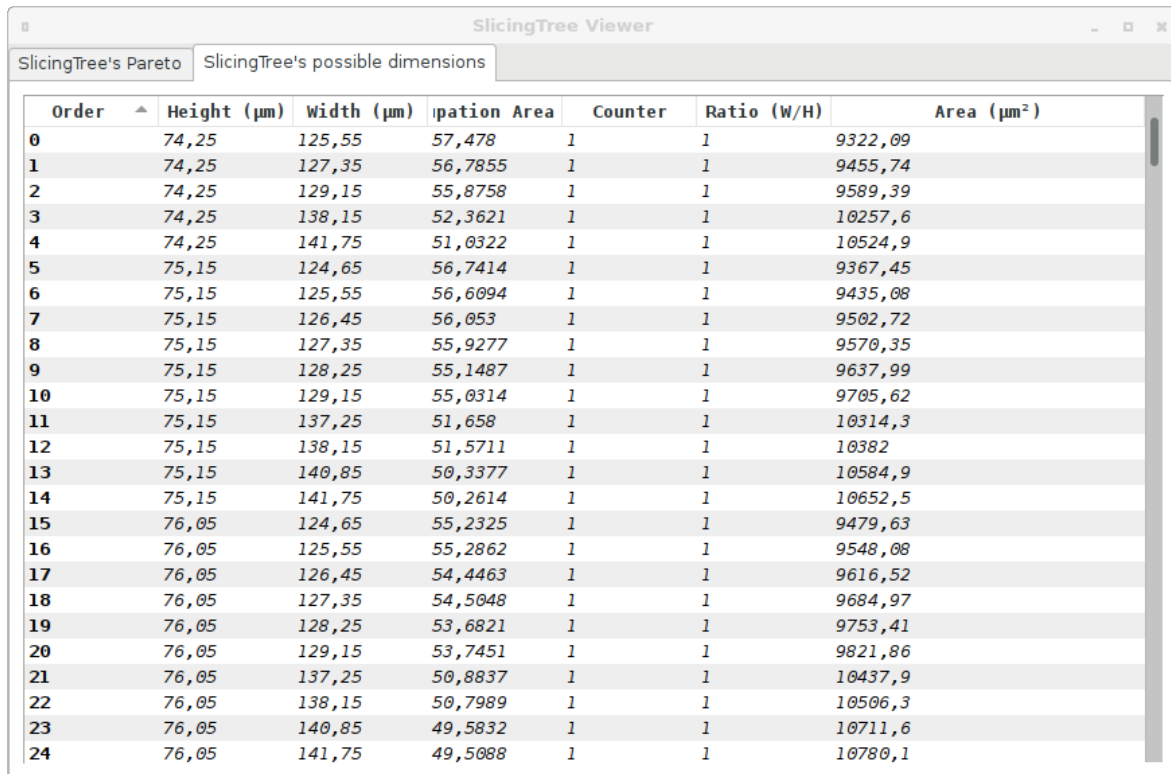


FIGURE 3.33 – Interface utilisateur - Visualisateur

Le tableau présentant tous les facteurs de forme possible est présenté par la figure 3.34. La ligne sélectionnée en bleu correspond aux informations du placement choisi. Les colonnes allant de gauche à droite correspondent à :

- **Order** : Le numéro du placement permettant de reproduire le placement à partir d'un script.
- **Height** : Hauteur du circuit en μ .
- **Width** : Largeur du circuit en μ .
- **Occupation Area** : Pourcentage d'occupation de l'espace des modules par rapport à la surface totale du circuit.
- **Counter** : Nombre de combinaisons différentes pour obtenir ce même facteur de forme.
- **Ratio H/W** : Rapport entre la hauteur et la largeur.



The screenshot shows a window titled "SlicingTree Viewer" with two tabs: "SlicingTree's Pareto" and "SlicingTree's possible dimensions". The active tab displays a table with the following columns: Order, Height (μm), Width (μm), Occupation Area, Counter, Ratio (W/H), and Area (μm²). The table contains 25 rows of data, with the last row (Order 24) highlighted in grey.

Order	Height (μm)	Width (μm)	Occupation Area	Counter	Ratio (W/H)	Area (μm²)
0	74,25	125,55	57,478	1	1	9322,09
1	74,25	127,35	56,7855	1	1	9455,74
2	74,25	129,15	55,8758	1	1	9589,39
3	74,25	138,15	52,3621	1	1	10257,6
4	74,25	141,75	51,0322	1	1	10524,9
5	75,15	124,65	56,7414	1	1	9367,45
6	75,15	125,55	56,6094	1	1	9435,08
7	75,15	126,45	56,053	1	1	9502,72
8	75,15	127,35	55,9277	1	1	9570,35
9	75,15	128,25	55,1487	1	1	9637,99
10	75,15	129,15	55,0314	1	1	9705,62
11	75,15	137,25	51,658	1	1	10314,3
12	75,15	138,15	51,5711	1	1	10382
13	75,15	140,85	50,3377	1	1	10584,9
14	75,15	141,75	50,2614	1	1	10652,5
15	76,05	124,65	55,2325	1	1	9479,63
16	76,05	125,55	55,2862	1	1	9548,08
17	76,05	126,45	54,4463	1	1	9616,52
18	76,05	127,35	54,5048	1	1	9684,97
19	76,05	128,25	53,6821	1	1	9753,41
20	76,05	129,15	53,7451	1	1	9821,86
21	76,05	137,25	50,8837	1	1	10437,9
22	76,05	138,15	50,7989	1	1	10506,3
23	76,05	140,85	49,5832	1	1	10711,6
24	76,05	141,75	49,5088	1	1	10780,1

FIGURE 3.34 – Interface utilisateur - Tableau des facteurs de forme

- **Area** : Occupation du circuit en μ^2 .

On peut utiliser le numéro de placement pour scripter un placement désiré. Connaissant ce numéro, on utilise la méthode *self.updatePlacement(numéro)* suite à la description du *slicing tree*. Une fois un placement choisi, le concepteur peut passer à l'étape suivante qui consiste à la phase de routage du circuit.

3.6 Conclusion

La phase de placement analogique et mixte est réalisée de manière assistée par le concepteur. Notre approche consiste à donner suffisamment de contrôle au concepteur afin de générer un placement respectant des contraintes décrites. En opposition aux placeurs de l'état de l'art qui se basent majoritairement sur des approches de recuit simulé, notre placeur est moins automatisé mais permet au concepteur de contrôler la topologie du circuit sans la perturber. Le concepteur a pour rôle de décrire la topologie du circuit sous la forme d'un *slicing tree* à partir d'un script python. La description du *slicing tree* et des contraintes spécifiées permet au placeur d'établir l'ensemble des facteurs de forme possible que peut prendre le circuit complet. Ces facteurs de forme sont présentés au concepteur sous la forme d'un nuage de points et d'un tableau interactif permettant de générer une solution de placement en l'espace de quelques secondes.

Une fois que le concepteur a choisi une solution de placement, la prochaine étape consiste à router le circuit mixte. Le routage mixte est divisé en deux étapes de routage : une phase de routage global et une phase de routage détaillé. Le résultat du placement est transformé en pavage d'espace pouvant soit représenter un espace de routage ou un module analogique ou numérique. La phase de routage global consiste à déterminer pour chacun des nets par quels pavés les fils de routage passeront et la phase de routage détaillé consiste à construire les fils de routage et à résoudre les problèmes de superpositions. Contrairement à la phase de placement où le placement numérique et analogique sont séparés, l'intégralité de la phase de routage est commune pour les *nets* analogiques et numériques. Le routeur global considère des contraintes supplémentaires ou des méthodes de calculs différentes en fonction du type du *net* traité.

3.7 Références

- [1] Christophe Alexandre. *Coriolis : une plate-forme ouverte pour l'évaluation de flots de conception VLSI fortement intégrés*. PhD thesis, Paris 6, 2007. [ix](#), [30](#), [32](#)
- [2] Stéphanie Youssef. *Designer-assisted, Reusable and Optimized Analog Layout Generation for Nanometric CMOS Era*. PhD thesis, Université Pierre et Marie Curie, (UPMC), December 2012. [32](#), [55](#), [62](#)
- [3] Nuno C. C. Lourenço Nuno C. G. Horta Ricardo M. F. Martins. *Generating Analog IC Layouts with LAYGEN II*. Springer Berlin Heidelberg, 2013. [ix](#), [34](#), [35](#)
- [4] Po-Hsun Wu, Mark Po-Hung Lin, Yang-Ru Chen, Bing-Shiun Chou, Tung-Chieh Chen, Tsung-Yi Ho, and Bin-Da Liu. Performance-driven analog placement considering monotonic current paths. In *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, pages 613–619. IEEE, 2012. [ix](#), [34](#), [38](#), [43](#)
- [5] Mark Po-Hung Lin, Hongbo Zhang, Martin DF Wong, and Yao-Wen Chang. Thermal-driven analog placement considering device matching. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(3) :325–336, 2011. [ix](#), [35](#), [41](#), [43](#)
- [6] Shigetoshi Nakatake. Structured placement with topological regularity evaluation. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 215–220. IEEE Computer Society, 2007. [35](#), [40](#)
- [7] Shigetoshi Nakatake, Masahiro Kawakita, Takao Ito, Masahiro Kojima, Michiko Kojima, Kenji Izumi, and Tadayuki Habasaki. Regularity-oriented analog placement with diffusion sharing and well island generation. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, pages 305–311. IEEE Press, 2010. [35](#), [40](#), [43](#)
- [8] Pang-Yen Chou, Hung-Chih Ou, and Yao-Wen Chang. Heterogeneous b*-trees for analog placement with symmetry and regularity considerations. In *Proceedings of the International Conference on Computer-Aided Design*, pages 512–516. IEEE Press, 2011. [ix](#), [36](#), [40](#), [41](#), [43](#)
- [9] F.Y. Young and D.F. Wong. Slicing floorplans with pre-placed modules. In *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on*, pages 252–258, 1998. [ix](#), [36](#), [38](#), [43](#)
- [10] Cheng-Wu Lin, Jai-Ming Lin, Chun-Po Huang, and Soon-Jyh Chang. Performance-driven analog placement considering boundary constraint. In *Proceedings of the 47th Design Automation Conference*, pages 292–297. ACM, 2010. [x](#), [36](#), [37](#), [41](#), [43](#)
- [11] D.W. Jepsen and C.D. Gellat Jr. Macro placement by monte carlo annealing. In *in Proc. IEEE Int. Conf. on Comp. Design*, pages 495–498, 1983. [37](#)
- [12] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *DAC '86 Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 101–107. IEEE, 1986. [38](#)
- [13] T. Abthoff and F. Johannes. Tina : Analog placement using enumerative techniques capable of optimizing both area and net length. In *EURO-DAC '96 with EURO-VHDL '96*, pages 398–403, 1996. [38](#), [43](#)

- [14] Juan A. Prieto, Adoracion Rueda, Jose M. Quintana, and Jose L. Huertas. A performance-driven placement algorithm with simultaneous place&route optimization and for analog and ic's. In *Proceedings of the 1997 European conference on Design and Test*, page 389. IEEE, 1997. [38](#), [43](#)
- [15] Fung Yu Young, DF Wong, and Hannah Honghua Yang. Slicing floorplans and with boundary and constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9) :1385–1389, 1999. [38](#), [43](#)
- [16] Fung Yu Young, DF Wong, and Hannah Honghua Yang. Slicing floorplans with range constraint. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(2) :272–278, 2000. [38](#), [43](#)
- [17] Mark Po-Hung Lin, Bo-Hao Chiang, Jen-Chieh Chang, Yu-Chang Wu, Rong-Guey Chang, and Shuenn-Yuh Lee. Augmenting slicing trees for analog placement. In *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2012 International Conference on*, pages 57–60. IEEE, 2012. [38](#), [43](#)
- [18] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. Vlsi module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12) :1518–1524, 1996. [x](#), [39](#), [40](#)
- [19] F. Balasa and K. Lampaert. Symmetry within the sequence-pair representation in the context of placement for analog design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19 :721–731, 2000. [40](#), [43](#)
- [20] Florin Balasa and Sarat C Maruvada. Using non-slicing topological representations for analog placement. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 84(11) :2785–2792, 2001. [40](#), [43](#)
- [21] Yiu-Cheong Tam, Evangeline FY Young, and Chris Chu. Analog placement with symmetry and other placement constraints. In *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, pages 349–354, 2006. [40](#), [43](#)
- [22] Long Di, Hong Xianlong, and Dong Sheqin. Signal-path driven partition and placement for analog circuit. In *Design Automation, 2006. Asia and South Pacific Conference on*, pages 6–pp, 2006. [40](#), [43](#), [51](#)
- [23] Linfu Xiao, Evangeline FY Young, Xiaoyong He, and Kong-Pang Pun. Practical placement and routing techniques for analog circuit designs. In *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, pages 675 – 679, 2010. [40](#), [43](#)
- [24] Florin Balasa. Modeling non-slicing floorplans with binary trees. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 13–16, 2000. [41](#), [43](#)
- [25] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-trees : a new representation for non-slicing floorplans. In *Proceedings of the 37th Annual Design Automation Conference*, pages 458–463, 2000. [41](#)

- [26] Florin Balasa, Sarat C Maruvada, and Karthik Krishnamoorthy. Efficient solution space exploration based on segment trees in analog placement with symmetry constraints. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 497–502, 2002. [41](#), [43](#)
- [27] Sarat C Maruvada, Ariel Berkman, Karthik Krishnamoorthy, and Florin Balasa. Deterministic skip lists in analog topological placement. In *ASIC, 2005. ASICON 2005. 6th International Conference On*, volume 2, pages 834–837. IEEE, 2005. [41](#), [43](#)
- [28] Martin Strasser, Michael Eick, Helmut Gräß, Ulf Schlichtmann, and Frank M Johannes. Deterministic analog circuit placement using hierarchically bounded enumeration and enhanced shape functions. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 306–313. IEEE Press, 2008. [41](#), [43](#)
- [29] Po-Hung Lin, Yao-Wen Chang, and Shyh-Chang Lin. Analog placement based on symmetry-island formulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(6) :791–804, 2009. [41](#), [43](#)
- [30] Hui-Fang Tsao, Pang-Yen Chou, Shih-Lun Huang, Yao-Wen Chang, Mark Po-Hung Lin, Duan-Ping Chen, and Dick Liu. A corner stitching compliant b*-tree representation and its applications to analog placement. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 507–511. IEEE, 2011. [41](#), [43](#)
- [31] Pei-Ning Guo, Chung-Kuan Cheng, and Takeshi Yoshimura. An o-tree representation of non-slicing floorplan and its applications. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 268–273. ACM, 1999. [42](#)
- [32] Yingxin Pang, Florin Balasa, Koen Lampaert, and Chung-Kuan Cheng. Block placement with symmetry constraints based on the o-tree non-slicing representation. In *Proceedings of the 37th Annual Design Automation Conference*, pages 464–467. ACM, 2000. [42](#), [43](#)
- [33] Linfu Xiao and Evangeline FY Young. Analog placement with common centroid and 1-d symmetry constraints. In *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pages 353–360. IEEE, 2009. [42](#), [43](#)
- [34] Jai-Ming Lin and Yao-Wen Chang. Tcg : a transitive closure graph-based representation for non-slicing floorplans. In *Proceedings of the 38th annual Design Automation Conference*, pages 764–769. ACM, 2001. [42](#)
- [35] Lihong Zhang, C-J Richard Shi, and Yingtao Jiang. Symmetry-aware placement with transitive closure graphs for analog layout design. In *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pages 180–185. IEEE, 2008. [42](#), [43](#)
- [36] Jai-Ming Lin, Guang-Ming Wu, Yao-Wen Chang, and Jen-Hui Chuang. Placement with symmetry constraints for analog layout design using tcg-s. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 1135–1137. ACM, 2005. [43](#)
- [37] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983. [43](#)

- [38] Stéphanie Youssef. *Aide au concepteur pour la génération de masques analogiques, réutilisables et optimisés, en technologie CMOS nanométrique*. PhD thesis, Université Pierre et Marie Curie-Paris 6, 2012. [50](#)
- [39] Po-Hsun Wu, Mark Po-Hung Lin, Tung-Chieh Chen, Ching-Feng Yeh, Tsung-Yi Ho, and Bin-Da Liu. Exploring feasibilities of symmetry islands and monotonic current paths in slicing trees for analog placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(6) :879–892, Jun 2014. [51](#)
- [40] Christian Masson Gabriel Gouvine Sophie Belloeil Damien Dupuis Christophe Alexandre Hugo Clement Marek Sroka Jean-Paul Chaput, Rémy Escassut and Wu Yi-fei. Coriolis, 2018. [61](#), [69](#)
- [41] Jacky Porte. *OCEANE : Outils pour la Conception et l'Enseignement des circuits intégrés ANalogiquEs – reference manual*, January 2018. [62](#)

Chapitre 4

Routage global

Sommaire

4.1 Introduction	78
4.2 Formalisation du problème du routage global	79
4.2.1 Le graphe de routage global	79
4.2.2 Contraintes du routage global	82
4.3 Les méthodes de résolution existantes	86
4.3.1 Les arbres de Steiner	86
4.3.2 La programmation linéaire	87
4.3.3 Routage par exploration exhaustive	88
4.4 Méthode de résolution	89
4.4.1 Fonction de coût	89
4.4.2 Construction d'arbres d'interconnexions en utilisant l'algorithme de Dijkstra	93
4.4.3 Algorithme de Dijkstra appliqué à la partie analogique du graphe de routage	98
4.5 Implémentation du routage mixte	103
4.5.1 Flot du routage global	104
4.5.2 Construction du graphe de routage global	105
4.5.3 Initialisation de l'algorithme de Dijkstra	109
4.5.4 Mise en oeuvre de l'algorithme de Dijkstra	112
4.5.5 Matérialisation des fils de routage global	122
4.5.6 Mise à jour de l'occupation des fils de routage et redimensionnement des canaux de routage	123
4.5.7 Le <i>slicing tree</i> durant la phase de routage global	124
4.5.8 Description des contraintes de routage dans le script <i>Python</i>	125
4.6 Conclusion	127
4.7 Références	128

4.1 Introduction

Considérant les modules d'un circuit placés, la phase de routage consiste à connecter électriquement l'ensemble des terminaux des modules à partir d'une *netlist* comprenant les informations de connexions du circuit. En d'autres termes, cela revient à déterminer pour chacun des *nets*, leurs niveaux de métaux, la position des *VIAs* ainsi que la position exacte des chemins du *net*. À la différence du routage pour les circuits numériques, le routage des circuits analogiques et mixtes doit réaliser cette tâche tout en tenant compte de plusieurs aspects :

- **Symétries** : Les contraintes de symétries sont particulièrement répandues. Dans le contexte des signaux différentiels, il est nécessaire de placer les fils de manière symétrique de telle sorte que le chemin des fils, la longueur des fils, la largeur des fils, le nombre de *VIAs* et le nombre de coudes soient identiques afin d'apparier les parasites issus du dessin des masques.
- **Topologies** : Pour garantir un environnement similaire à certaines parties du circuit, des contraintes de topologies sont parfois requises. À la différence d'une symétrie, il s'agit de placer des fils de routage en suivant une topologie avec le même nombre de fils et de direction de fils sans pour autant qu'il y ait une symétrie parfaite par rapport à un axe.
- **Densité de courant** : Les densités de courant présentes dans les circuits analogiques et mixtes de différentes portions du circuit peuvent être extrêmement différentes. Avec l'intégration des circuits suivant des gravures de plus en plus fines, l'électromigration dans les fils à fort courant nécessite de contrôler leur largeurs. La capacité à gérer des fils de largeurs différentes est donc importante.
- **Diaphonie** : La diaphonie entre plusieurs *nets* peut engendrer de fortes dégradations des performances d'un circuit. Pour résoudre ce type de problème, la solution la plus commune consiste à utiliser des anneaux de garde ou le routage canal permettant l'isolation de certaines portions sensibles du circuit.

La qualité du routage a un impact conséquent sur les performances d'un circuit. La capacité à pouvoir gérer les contraintes de routage mentionnées précédemment est considérée par les différentes méthodes de résolution de l'état de l'art qui seront présentées dans la section 4.3. Dans le cadre de notre environnement, nous avons opté pour le choix d'une stratégie avec un routage en deux étapes similaires au routage numérique (routage global et détaillé).

Dans le cadre des circuits mixtes, il est nécessaire de tenir compte tant des circuits numériques que des circuits analogiques. Par conséquent, le routeur doit être en mesure de gérer simultanément les interconnexions entre des modules numériques et analogiques en tenant compte des contraintes de chacun des deux types de circuits. Pour ce faire, un des aspects majeurs de notre approche est d'avoir une structure de données uniforme s'appliquant d'une part entre les parties numériques et analogiques et d'autre part entre les étapes de placement et de routage du flot de conception. Cette uniformité s'applique au niveau du routage dans la mesure où l'algorithme employé est le même pour les deux types de circuits avec pour différence que les contraintes sont adaptées en fonction du circuit traité. Des considérations supplémentaires sont également prises en compte pour le circuit analogique, telle que la gestion de fils symétriques.

Notre approche de routage mixte est composée de deux étapes : une étape de routage global et de routage détaillé. Le routage global se base sur une méthode d'exploration utilisant l'algorithme de Dijkstra pour trouver la meilleure interconnexion à travers les découpes du *slicing tree* résultant d'un placement. Les contraintes employées lors de l'estimation des chemins optimaux sont choisies en fonction du type de modules à interconnecter. Suite au routage global, les fils de routage se retrouvent délimités au sein de canaux de routage dans lequel ils doivent être placés. Dans la phase de routage détaillé, les fils sont créés avec la topologie adaptée afin de les faire passer par les canaux de routage déterminés par la phase de routage global. Le placement de ces fils sans superposition est réalisé à travers un algorithme de *rip-up and reroute* à fortes contraintes pour les *nets* critiques.

Dans ce chapitre, nous présentons dans un premier temps le problème du routage global avec sa modélisation à partir de structures de graphe de routage permettant de représenter les ressources disponibles sur chacune des parties d'un circuit. Dans un second temps, nous établissons les différents critères qui définissent la qualité d'une solution de routage global. Ensuite, nous présentons des méthodes de résolution usuelles de l'état de l'art ainsi que la méthode employée dans le cadre de notre routeur global. Les détails concernant cette méthode de résolution sont présentés avec l'influence que peuvent avoir les différentes contraintes sur l'algorithme de recherche de chemins optimaux. Nous décrivons enfin la mise en œuvre ainsi que la structure de données destinée à l'entière phase de routage traitant de manière uniforme les circuits numériques et analogiques.

4.2 Formalisation du problème du routage global

La résolution du problème du routage global se base sur l'utilisation d'un graphe de routage. Il permet de représenter les ressources ainsi que la topologie dédiée à l'espace de routage de manière épurée. Cette modélisation des ressources de routage a un impact sur l'efficacité des algorithmes en terme de performances. En particulier, un graphe de routage chargé excessivement en données peut s'avérer coûteux en ressource de calcul. Une représentation de l'espace de routage trop fine en granularité peut entraîner la création de longs arbres d'interconnexions en termes de nombre de modules à parcourir, cela peut engendrer un long temps d'exécution et être coûteux en ressource mémoire.

Dans cette section, nous présentons le graphe de routage que nous employons dans le contexte du routage des circuits mixtes. Puis nous détaillons l'ensemble des critères jouant sur la qualité d'une solution de routage.

4.2.1 Le graphe de routage global

Définition du graphe de routage

Suite à la phase de placement, l'ensemble du circuit est transformé en une représentation sous forme de grille ce qui est communément utilisé par les routeurs de l'état de l'art. Ce pavage peut être régulier ou non régulier selon l'approche utilisée. Dans notre cas, nous utilisons une grille de routage régulière pour les surfaces contenant uniquement des modules numériques et une grille non régulière pour les surfaces du circuit contenant des

modules analogiques.

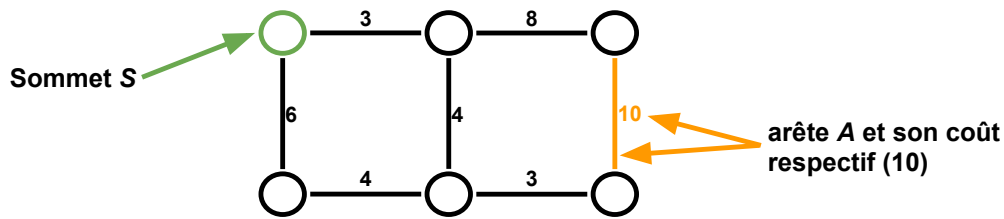


FIGURE 4.1 – Exemple de graphe de routage global $G(S, A)$

On appelle $G(S, A)$ le graphe de routage ainsi obtenu dans lequel chacun des pavages obtenus est représenté par un sommet S et leurs connexités sont représentées par des arêtes nommées A (voir figure 4.1. Notons également :

- Il existe un sommet $s_i \in S$ avec $i \in \{1, \dots, |S|\}$ pour un pavé du circuit, que ce soit un espace représentant un module ou un espace de routage.
- Il existe une arête $a_{ij} \in A$ reliant deux sommets s_i et s_j s'ils sont associés à deux pavés adjacents et qu'il existe suffisamment de place pour considérer une piste de routage permettant de passer de l'un à l'autre.

Le graphe de routage $G(S, A)$ obtenu à partir de la topologie du circuit est un graphe connexe non orienté, cela implique qu'il existe toujours un chemin, autrement dit une succession d'arêtes, pouvant connecter deux sommets s_i et $s_j \in S$ quelconques du graphe.

Chacune des arêtes est définie par :

- **Une capacité** : Elle représente le nombre maximum de pistes de routage pouvant passer entre les deux sommets. Cela dépend de la longueur de la partie commune entre deux pavés. On définit une piste de routage comme étant une position qu'un fil de routage peut prendre en respectant les règles de dessins avec son environnement (voir figure 4.2). Dans la mesure où la partie commune entre deux sommets adjacents ne permet pas d'avoir suffisamment d'espace pour laisser place à une piste de routage, l'arête sera alors toujours considérée bloquée. Notons que la capacité d'une arête tient compte de la capacité en nombre de pistes de routage pour toutes les couches de métal utilisables dans le cadre du routage. Cette capacité est définie lors de la création du graphe de routage et reste fixe.
- **Une occupation** : Elle indique le nombre de fils de routage déjà évalués passant par cette arête. Cela permet en particulier l'évaluation du niveau de congestion passant par cette arête. Plus le niveau de congestion est haut, plus on évite de passer par cette arête afin d'obtenir une meilleure occupation des arêtes du circuits et ainsi une meilleure répartition des fils de routage à travers les différents "chemins" du circuit. La gestion de la congestion est surtout utilisée pour le routage numérique qui nécessite de router un grand nombre de fils. La congestion est également un indicateur utilisé pour les parties analogiques dont l'utilisation sera présentée dans les sections suivantes.
- **Une longueur** : Elle définit le coût en distance pour passer d'un sommet à un autre. En principe, la longueur d'un chemin est égale à la somme des longueurs des arêtes

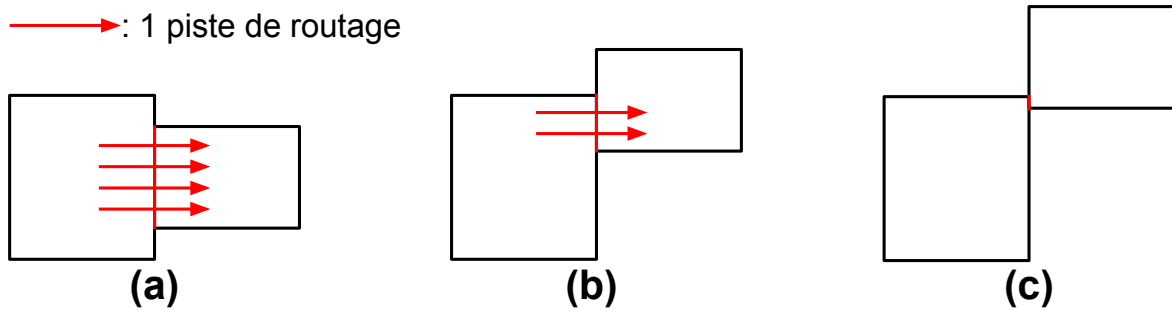


FIGURE 4.2 – (a) Arête avec une capacité de quatre pistes de routage (b) Arête avec une capacité de deux pistes de routage (c) Manque d'espace pour une piste de routage bien que les pavés soient adjacents.

parcourues. Cependant la méthode de calcul de longueur de chemin est différente en fonction du type de circuit considéré. Dans le cadre des circuits numériques et de leurs pavés réguliers, la longueur de l'arête est définie par la distance centre à centre entre deux pavés. De par leur topologie de pavages différent, la longueur d'arête d'un circuit analogique est évaluée de manière plus précise en estimant la distance entre le point d'entrée du pavé courant et le point de sortie vers le pavé suivant (voir figure 4.3) et plus de détails dans la partie 4.4.3 et 4.5.4). Les détails d'évaluation des distances sont présentés dans les sections suivantes. Que ce soit pour les circuits numériques ou analogiques, les distances entre deux points sont calculées en distance Manhattan.

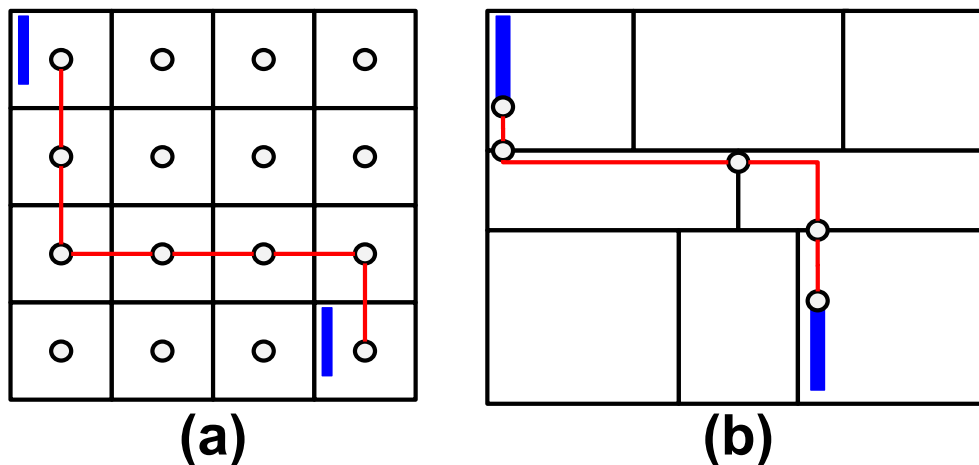


FIGURE 4.3 – Estimation de longueur d'un chemin dans un circuit numérique (a) et dans un circuit analogique (b)

Dans le processus d'estimation de longueurs de chemins, les arêtes peuvent être contrôlées afin de restreindre le passage de fils dans certains pavés de la grille. Par exemple, il est par moment nécessaire d'éviter certains obstacles à un niveau de couche de métal particulier rendant la traversée d'un pavé impossible. En analogique, il est impératif d'éviter de placer les fils de routage au-dessus des zones actives des transistors, cela peut parasiter le fonctionnement des transistors. Par conséquent, l'ensemble des pavés représentant un *device*, ne faisant pas partie du *net* à router, seront invalidés afin de préserver leur bon fonctionnement.

La figure 4.4 montre deux exemples de graphe de routage, pour un circuit numérique et un circuit analogique. Le circuit numérique est découpé selon un pavage régulier (grille verte) tandis que le pavage du circuit analogique est irrégulier et dépend directement de la forme des modules. Les sommets sont représentés par les points et les arêtes par les lignes connectant deux points.

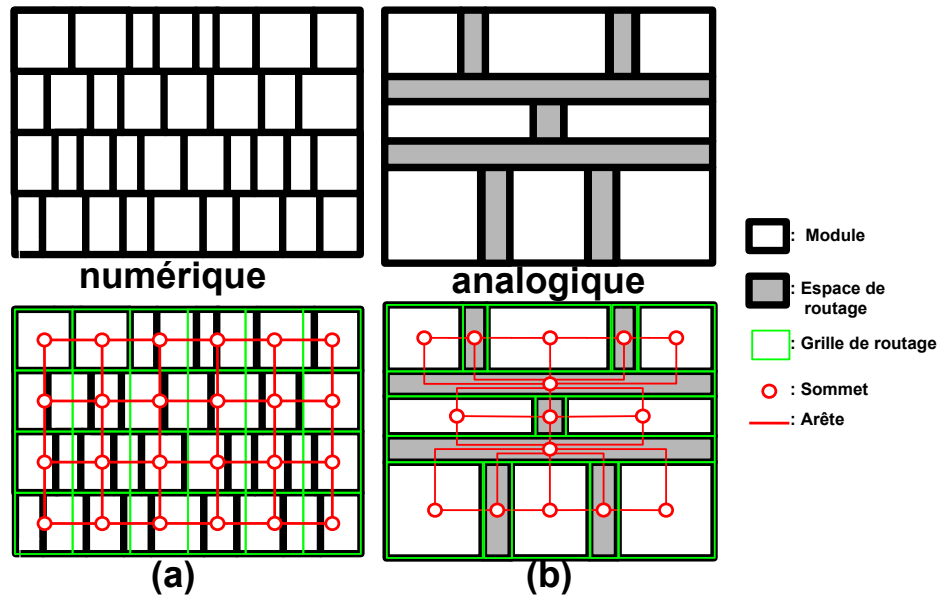


FIGURE 4.4 – Graphe de routage d'un circuit numérique (a) et d'un circuit analogique (b)

Représentation d'un *net* sur le graphe de routage

Le routage d'un *net* est représenté par un sous-graphe connexe rejoignant les sommets des pavés contenant les connecteurs du *net* et ses interconnexions (voir figure 4.5). Au cours de la recherche de la solution du routage global, le sous-graphe représentant le *net* s'étendra jusqu'à contenir l'ensemble des connecteurs. On considère le routage global achevé lorsque chacun des *nets* est représenté par un sous-graphe rejoignant leurs connecteurs.

Pour chacun des *nets*, la position du connecteur définit le sommet auquel il sera associé. Dans notre approche avec les circuits analogiques, les connecteurs sont toujours contenus dans un pavé unique. Pour les circuits numériques, le connecteur peut par moment se trouver sur plusieurs pavages différents. Par conséquent, on considère le connecteur associé à l'ensemble des sommets dans lesquels il se trouve. Lorsque l'ensemble des connecteurs d'un *net* se trouve au sein d'un même pavé, le *net* se limite au sommet contenant les connecteurs.

4.2.2 Contraintes du routage global

Contraintes de diaphonie

Le temps de propagation d'un signal électrique à travers un fil d'interconnexion est proportionnel à la longueur du fil, plus ce dernier est long, plus le temps de propagation

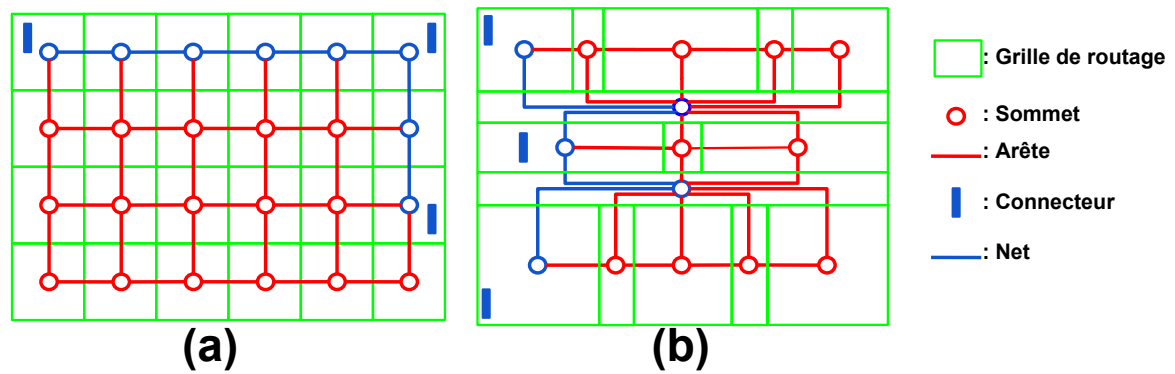


FIGURE 4.5 – Représentation d'un *net* sur le graphe de routage d'un pavage numérique (a) et d'un pavage analogique (b)

le sera. La longueur des fils d'interconnexion est un critère déterminant les performances d'un circuit. Avec les technologies nanométriques, les fils longs peuvent poser des problèmes de bruit ou d'effet d'antenne. Lorsque deux fils d'interconnexion restent voisins sur une trop grande longueur, un effet capacitif (*crosstalk*) perturbe les signaux de chacun. Il est donc nécessaire de minimiser la longueur totale des fils d'interconnexion. Dans le contexte du routage global, cela revient à faire en sorte de minimiser les arbres d'interconnexion à une longueur minimale.

La réduction de la longueur totale des fils d'interconnexions permet de diminuer les temps de propagation dans les fils et de réduire les bruits parasites. Néanmoins, la restriction à des fils courts uniquement peut conduire à une augmentation de la congestion. Les arbres d'interconnexion entre deux composantes connexes deviennent plus directs, avec peu ou pas de détours, ce qui peut créer des zones congestionnées.

Minimisation du nombre de VIAs

On cherche également à minimiser les *VIAs* dans le cadre du routage. De par leur nature résistive, ils entraînent une dégradation des performances du circuit. De plus pour les technologies les plus récentes, il est requis que ces derniers doivent être dupliqués, cela implique qu'au minimum deux *VIAs* doivent être placés côte à côte. Ce dédoublement occupe alors plus de ressources, obstruant plusieurs pistes de routage pour certaines couches de métal. Minimiser le nombre de *VIAs* dans le contexte du routage global consiste à réduire le plus possible le nombre de changements de direction des fils de routage.

Sur la figure 4.6, on observe deux chemins connectant une source, notée "S", à une destination, notée "D", de façon différente et ces chemins sont de longueurs identiques. En considérant que chacune des arêtes coûte une longueur de 1, chacun des chemins a une longueur de 7. Le chemin (a) contient un changement de direction et par conséquent comporte un via. En revanche, le chemin (b) comporte 6 changements de direction et donc six *VIAs*. Le chemin (a) est donc préférable au chemin (b).

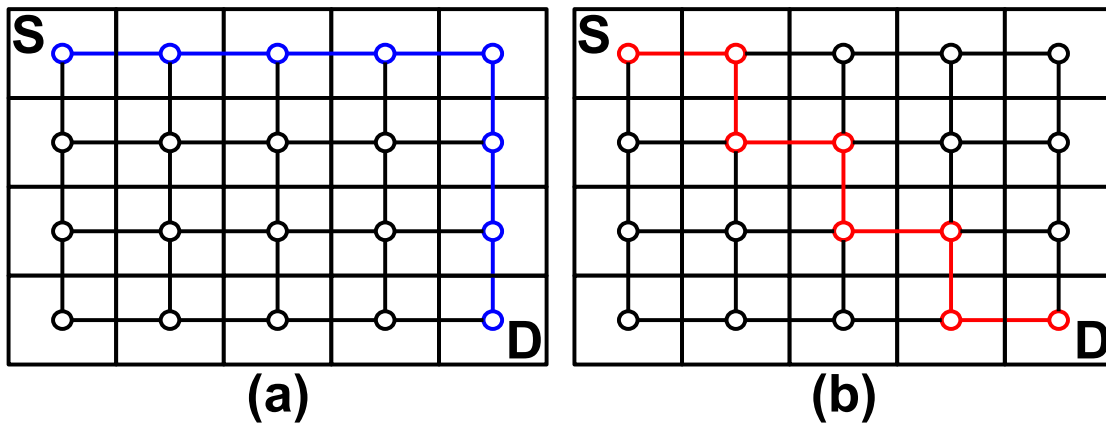
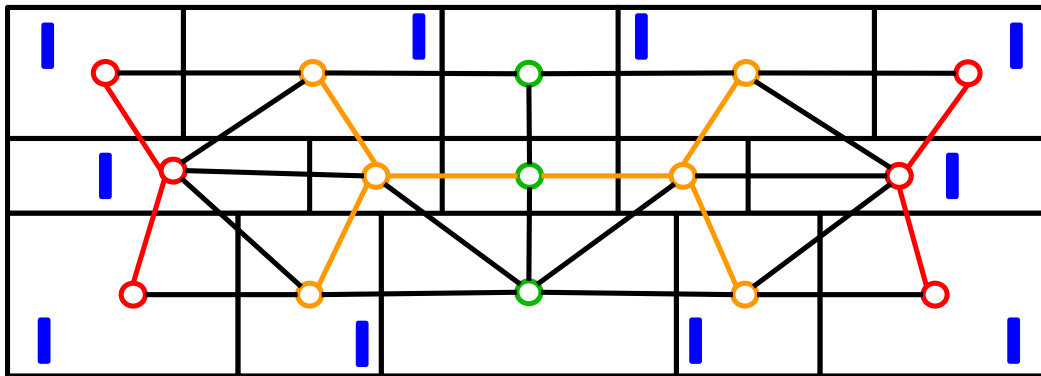


FIGURE 4.6 – Chemins connectant une source et une destination

FIGURE 4.7 – Représentation de deux symétries : une paire de *nets* symétriques en rouge et un *net* dont les sommets sont symétriques en orange. Les sommets des pavés se trouvant sur l'axe de symétrie de ces deux *nets* sont représentés en vert

Routage symétrique

Comme mentionné dans le chapitre précédent, les contraintes de symétrie sont couramment utilisées dans le contexte des circuits analogiques. Par conséquent, il est nécessaire de les considérer lors de la phase de routage et d'être capable de les exprimer au sein du graphe de routage.

D'un point de vue routage global, obtenir un routage symétrique consiste à trouver deux sous-graphes symétriques en terme de parcours de sommet de même coût par rapport aux sommets situés au niveau de l'axe de symétrie. Pour cela, la recherche de l'arbre d'interconnexions d'un *net* symétrique est réalisée sur la moitié du graphe délimitée par les sommets de référence se trouvant au niveau de l'axe de symétrie.

On peut aussi être amené à considérer un *net* unique symétrique par rapport à un axe de symétrie horizontal ou vertical. Tout comme avec deux paires symétriques, on se restreint à la moitié du graphe de routage délimité par les sommets se trouvant au niveau de l'axe de symétrie. L'arbre d'interconnexions à trouver est composé non seulement des sommets présents sur la moitié du graphe mais également d'un des sommets au niveau de l'axe de symétrie.

Largeur de fils variable

Les aspects d'électromigration sont des problèmes particulièrement importants en particulier avec l'évolution de l'intégration des circuits intégrés. Afin de réduire les risques d'électromigration, la largeur de chacun des fils et le nombre de *VIA*s doivent être déterminés au préalable en fonction de l'intensité maximale passant par ce *net*. Cela implique que les fils de routage seront un multiple de pistes de routage et qu'il est nécessaire de gérer l'occupation en pistes de routage pour chacune des arêtes lors de la recherche d'arbre d'interconnexions.

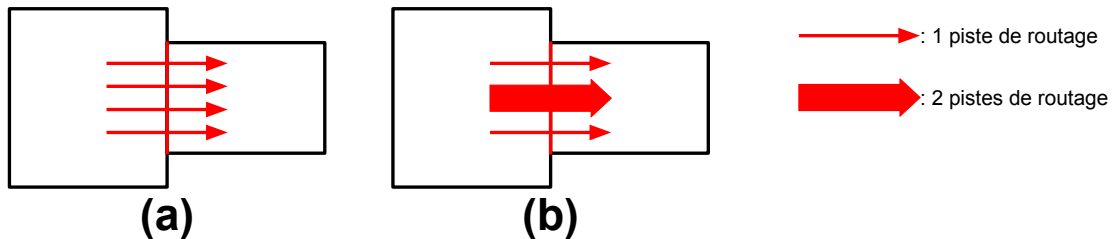


FIGURE 4.8 – Occupation de pistes en fonction de la taille des fils. (a) est occupé par des fils avec un coût d'une 1 piste chacun (b) contient un fil avec une occupation de deux pistes de routage.

Restriction d'utilisation de sommets

De manière générale, le routage des fils des circuits numériques peut passer sur l'espace dédié au circuit, en particulier au dessus des cellules numériques sans restriction particulière (*routing-over-the-cells*). En revanche, il est interdit d'avoir des fils passant au-dessus des modules analogiques critiques car cela peut parasiter leur fonctionnement. Par conséquent, certains sommets du circuit sont restreints et rendus inutilisables pour le routeur en dehors du *net* concerné par le sommet. L'exemple de la figure 4.9 montre les sommets restreints (en rouge) que le routage global ne peut utiliser.

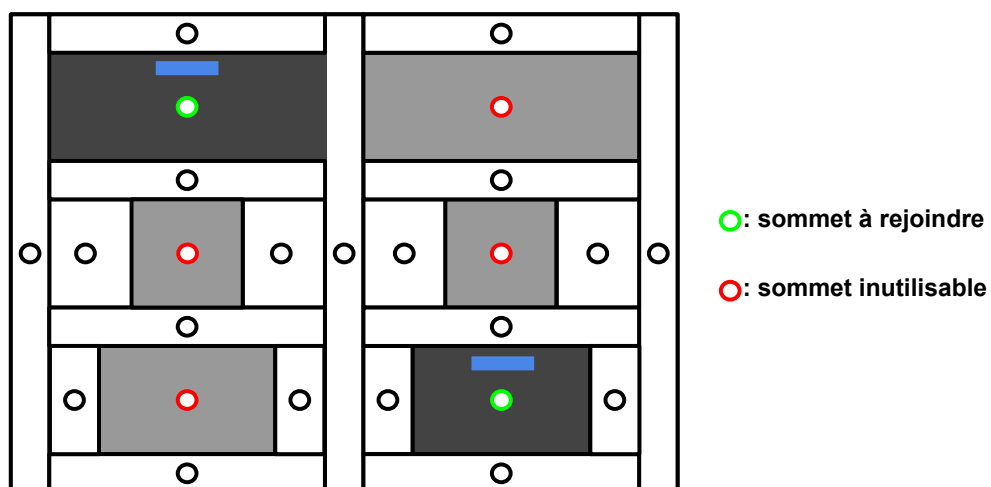


FIGURE 4.9 – Restriction d'accès de sommets du graphe de routage

4.3 Les méthodes de résolution existantes

Le problème du routage consiste à définir l'ensemble des chemins d'interconnexions des *nets* en minimisant les longueurs de fils tout en tenant compte des contraintes de routage analogiques et mixtes mentionnées précédemment. Dans le cadre des circuits numériques, le problème se présente sous un grand nombre d'interconnexions à trouver soumises à de faibles contraintes tandis que pour les circuits analogiques et mixtes, le problème se présente comme un petit nombre d'interconnexions soumis à des contraintes restrictives. Cette section 4.3 présente des approches de routage utilisées dans l'état de l'art du routage pour circuits analogiques et mixtes.

4.3.1 Les arbres de Steiner

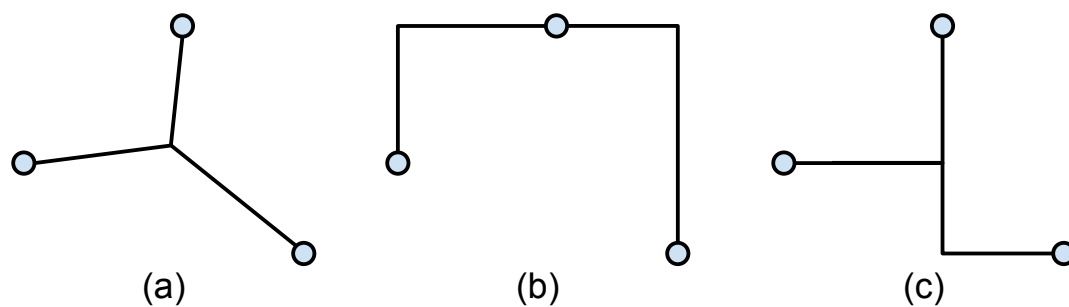


FIGURE 4.10 – (a) Arbre minimal de Steiner, (b) arbres de Steiner minimal rectilinéaire et (c) un arbre couvrant

Les arbres de Steiner sont communément employés pour résoudre des problèmes de recherche d'interconnexions de plusieurs points. Un arbre minimal de Steiner (voir 4.10.(a)) définit un arbre d'interconnexions d'un *net* avec une longueur minimale de fil. Dans les limites des circuits intégrés et l'usage de fils uniquement horizontaux et verticaux, il est commun d'utiliser un arbre de Steiner minimal rectilinéaire (voir 4.10.(b)) avec des connexions de terminal à terminal. Il existe également la représentation sous forme d'arbre couvrant de poids minimal (voir 4.10.(c)), sans contrainte de connexion de terminal à terminal.

Une méthode usuelle de construction d'arbre de Steiner consiste à considérer les sommets à relier comme des points d'un plan. Ce plan peut être défini par une grille représentant la grille de routage, c'est-à-dire les endroits sur lesquels un fil peut être placé. Des obstacles peuvent être représentés afin d'éviter certaines zones du circuit. On utilise alors un graphe de recherche de chemin pour la construction d'arbres de Steiner composant un ensemble de sommets d'arc les rejoignant (voir figure 4.1). La distance séparant deux nœuds correspond au coût de passage d'un nœud à l'autre. Il est commun de considérer une fonction de coût faisant intervenir des contraintes diverses comme par exemple des contraintes de congestion. En fonction du coût, le graphe de routage devient virtuellement déformé pour que le calcul de la longueur des arbres prenne en compte les contraintes.

Il existe des outils permettant de construire des arbres rectilinéaires de longueur proche de l'optimal en un temps logarithmique en fonction du nombre de sommets à relier. L'outil *FLUTE*[1] permet de créer des arbres de Steiner allant jusqu'à 9 connecteurs et est couramment utilisé par exemple par Mohammad Torabi et al.[2] [3]. Jin-Tai Yan et al.[4][5] utilisent des arbres de Steiner en considérant les contraintes d'électromigration dans la fonction de coût afin de minimiser la longueur totale de fil tout en satisfaisant les lois de courant de Kirchhoff. Changxu Du et al.[6] et Hung-Chih Ou et al.[7] utilisent également des arbres de Steiner dans leur approche de routage.

Les arbres de Steiner proposent une solution optimale en terme de distance pour des *nets* à plusieurs connecteurs. D'autres facteurs sont également à prendre en compte tel que le nombre de *VIA*s, les appariements de topologies ou le nombre de changements de direction d'un fil.

4.3.2 La programmation linéaire

Une autre méthode de résolution du problème du routage consiste à utiliser la programmation linéaire en nombres entiers (*Integer Linear Programming - ILP*). Cette approche consiste à modéliser le problème du routage en un problème mathématique en considérant un ensemble d'équations linéaires traduisant les relations des variables et des contraintes d'un circuit à router. Suite à l'ordonnancement des *nets*, les méthodes de routage séquentielles prennent des décisions pour un *net* donné en se basant sur l'occupation de l'espace précédemment routé et ne tiennent pas compte des *nets* routés plus tard. Cela peut se traduire par une dégradation de la congestion ainsi que de la qualité de la gestion des contraintes de symétries, d'appariement de topologies et d'appariement de longueur de fils en même temps. C'est avec l'optique de considérer simultanément différentes contraintes, que l'*ILP* est employée.

L'utilisation de l'*ILP* se déroule en plusieurs étapes qui sont les suivantes :

- **Choix d'un modèle de grille :** Il est habituel d'utiliser des représentations de l'espace basées sur une grille afin de représenter les relations de distance et les chemins des fils de routage. La plus simple des grilles est la représentation d'une grille uniforme dans laquelle chacune des graduations est régulière. Cependant, le routage basé sur des grilles uniformes tend à être coûteux en terme d'occupation mémoire dans le cas des grilles à petites unités. Il existe également différentes formes de pas de grille comme par exemple, Ou Hung-Chih et al.[8] utilisent une grille octogonale afin de représenter des fils à 45 degrés qui peuvent être utilisés pour des raisons pratiques pouvant réduire la longueur de fils, les effets parasites et la consommation.
- **Formulation du problème, des variables et des contraintes :** Cette étape consiste à déterminer l'ensemble des modules placés, des contraintes affectées à chacun des *nets* (symétries, longueur de fils, topologies, ...) et les paramètres employés pour indiquer les terminaux d'un *net*.
- **Formulation de l'objectif et résolution :** Une fois l'ensemble de paramètres déterminés, il est nécessaire de formuler mathématiquement l'objectif à respecter. Cela revient de manière générale à minimiser la longueur de fils tout en respectant plusieurs contraintes. Les problèmes d'*ILP* sont ensuite résolus par des solveurs comme *IBM ILOG CPLEX Optimizer* [9].

Ou Hung-Chih et al.[8] sont parmi les premiers à introduire l'*ILP* pour résoudre le problème de routage. En plus de l'utilisation de la grille octogonale, ils effectuent le routage à travers plusieurs niveaux de précision en terme de description de l'espace et des modules à router. Wu Chia-Yu et al.[10] effectuent le routage en deux temps par une évaluation de chemins possibles avec une approche séquentielle dans un premier temps et par l'*ILP* pour déterminer les meilleurs chemins à prendre. Mohammad Torabi et al.[2][3] utilisent également l'*ILP* en routant les régions du circuit de manière hiérarchique.

La programmation linéaire a l'avantage de pouvoir gérer l'ensemble des *nets* simultanément permettant de gérer plus efficacement les problèmes de gestion de congestion des fils de routage. Il est également facile de gérer plusieurs contraintes en même temps lors de la recherche de solutions.

4.3.3 Routage par exploration exhaustive

Chin Yang Lee[11] introduit le principe des méthodes de routage par exploration exhaustive (*maze routing*). Ces méthodes consistent à considérer tous les chemins possibles reliant un sommet *source* à un sommet *destination* du graphe de routage. Ces méthodes démarrent au niveau d'un nœud source à partir duquel on évalue les distances parcourues vers les nœuds voisins. L'évaluation se propage ensuite vers les nœuds voisins de la même façon qu'une propagation d'onde à la surface de l'eau autour du nœud source jusqu'à atteindre le nœud destination. Cette propagation à travers le graphe peut être limitée par une fenêtre d'exploration permettant de restreindre l'algorithme à une portion du graphe contenant au minimum le nœud source et destination. Les méthodes d'exploration exhaustive se basent sur l'algorithme de Dijkstra [12] et garantissent de trouver le chemin de coût minimal s'il existe.

6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
5	4	3	2	3	4	X	X	X	10	11	12	13	14	15	16	17	18
4	3	2	1	2	3	4	5	X	11	12	13	14	15	16	17	18	19
3	2	1	S	1	2	3	4	X	X	X	X	X	X	17	18	19	20
4	3	2	1	2	3	4	5	X	23	22	21	20	19	18	19	20	21
5	4	3	2	3	4	5	6	X	24	23	22	21	20	19	20	21	22
6	5	4	3	4	5	6	7	X	23	22	D	20	21	20	21	22	23
7	6	5	4	5	6	7	8	X	22	21	20	19	18	19	20	21	22
8	7	6	5	6	7	8	9	X	21	20	19	18	17	18	19	20	21
9	8	7	6	7	8	9	10	X	X	X	X	17	20	21	22	23	24
10	9	8	7	8	9	10	11	12	13	14	15	16	19	20	21	22	23
11	10	9	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

FIGURE 4.11 – Exemple d'exploration exhaustive pour un *net* comprenant une source *S*, une destination *D* et un obstacle (*X*)

La figure 4.11 montre un exemple de résultat d'exécution d'un algorithme de Dijkstra. Dans cette grille, on peut se déplacer que de façon verticale et horizontale et on peut y observer :

- une source notée *S*,

- une cible notée D ,
- des obstacles notés X ,
- le coût en distance pour atteindre une case donnée à partir de la source.

Cet exemple utilise une grille uniforme dans laquelle le coût pour aller d'un nœud à un autre ne coûte que 1. Dans des cas pratiques, la grille peut être non uniforme et le coût d'une arête peut être différent de 1 et toujours supérieur à 0.

L'avantage d'une telle approche est de toujours être capable de trouver le chemin de coût minimal. Elle est également simple à implémenter et l'ajout de contraintes peut être réalisé en considérant une fonction de coût augmentant virtuellement les distances vers les sommets. Du fait de son approche exploratoire, elle peut être coûteuse en ressources mais cela peut être limité par l'utilisation de la fenêtre d'exploration.

4.4 Méthode de résolution

Dans cette section, nous présentons la méthode de résolution choisie dans le cadre de nos outils de routage pour circuits mixtes. Ce choix se base sur l'évaluation d'un compromis entre le problème du routage numérique et analogique et la recherche exhaustive nous semble la plus appropriée pour résoudre ce problème. Les arbres de Steiner sont adaptés uniquement pour le problème du routage numérique, le problème du routage analogique impose la considération d'obstacle rendant les arbres de Steiner difficilement utilisables. La programmation linéaire est raisonnablement utilisable uniquement pour le problème du routage analogique, le problème du routage numérique présente un trop gros volume de cellules à gérer.

L'exploration exhaustive est adaptée à la recherche de chemins optimaux et à la gestion d'obstacles qui s'avèrent être indispensables dans le cadre du routage analogique. L'utilisation de fenêtre d'exploration peut limiter la consommation en terme de ressources pour son utilisation avec le routage numérique mais également dans le cadre du routage analogique lors du routage de *net* symétrique. C'est pourquoi nous faisons le choix d'utiliser une approche pouvant résoudre les deux problèmes de routage numérique et analogique.

Nous définissons la fonction de coût prenant en compte les contraintes citées dans les précédentes sections et utilisées pour construire les arbres d'interconnexions optimaux. Par la suite, nous détaillons le fonctionnement de l'algorithme de Dijkstra employé pour déterminer le routage global. Afin de comprendre ses détails, nous présentons étape par étape l'algorithme de base, sa version pour traiter les nets à plus de deux points ainsi que la gestion de composantes multiples.

4.4.1 Fonction de coût

Une solution de routage global est caractérisée par un coût représentant la somme de l'ensemble des coûts des arbres d'interconnexions obtenus pour cette solution. Chaque arbre d'interconnexions est défini par un coût étant égal à la somme des coûts de chacune de ses arêtes qui elles-mêmes sont caractérisées par les critères suivants. Soit une solution R résultant de n arbres d'interconnexions A_i avec $i \in \{1, \dots, n\}$ et donc chacun des arbres

est composé de m arêtes a_j avec $j \in \{1, \dots, m\}$, le coût de cette solution se présente sous la forme suivante :

$$\begin{aligned} \text{cout}(R) &= \sum_{i \in \{1, \dots, n\}} \text{cout}(A_i) \\ \text{cout}(A_i) &= \sum_{j \in \{1, \dots, m\}} \text{cout}(a_j) \\ \text{cout}(a_j) &= \text{coutLongueur}(a_j) + \text{coutVias}(a_j) + \text{coutCongestion}(a_j) \end{aligned} \quad (4.1)$$

Chacun de ces critères est défini de façon différente en fonction du type de la zone à router. Pour des raisons de clarté, la gestion des parties numériques est résumée, davantage de détails peuvent être trouvés dans la thèse de Damien Dupuis[13]. Les paragraphes suivants détaillent en particulier la gestion de la fonction de coût des fils d'interconnexion, du nombre de VIAs et de la congestion pour la partie analogique.

Longueur totale des fils d'interconnexion numériques et analogiques

La longueur d'une arête correspond à la distance séparant deux sommets adjacents du graphe de routage. Soit deux sommets $S1_{(x_1, y_1)}$ et $S2_{(x_2, y_2)}$ de pavés numériques, la longueur de l'arête a reliant $S1$ et $S2$ est estimée par :

$$\text{cout}(a_j) = \text{Longueur}_{S1-S2} = |x_2 - x_1| + |y_2 - y_1| \quad (4.2)$$

(x_1, y_1) étant la position du centre du pavé du sommet $S1$, respectivement (x_2, y_2) pour $S2$. Ainsi, la longueur totale des fils d'un arbre d'interconnexions revient à calculer la somme des distances Manhattan entre les sommets composant l'arbre. La longueur totale des fils d'interconnexions pour une solution de routage global donnée est estimée par la somme des longueurs de fils des arbres d'interconnexion.

Pour la longueur d'une arête entre deux sommets représentant des pavés de circuits analogiques ou mixtes, il est nécessaire d'estimer la distance de façon différente principalement à cause de l'irrégularité du pavage analogique. Le pavage du circuit numérique est représenté sous forme d'une grille régulière, cela implique que tous les pavés numériques soient de même taille et de forme carrée. Ainsi, l'estimation de longueur d'arête utilisant la distance centre à centre des sommets donnera un résultat proche de la longueur de fils réels.

En comparaison, le pavage analogique est très irrégulier avec des pavés de forme rectangulaire et de tailles très différentes. Nous voulons montrer qu'utiliser l'estimation de longueur des sommets numériques pour des sommets analogiques peut conduire à des erreurs significatives. La figure 4.12 illustre la longueur de fil estimée (en bleu), on remarque qu'elle est erronée. Ces erreurs d'estimations peuvent entraîner des erreurs de recherche du chemin de coût minimal dans la construction des arbres d'interconnexion.

On doit tout de même considérer un coût en longueur pour chaque arête du graphe de routage de la partie analogique d'un point de vue théorique. À la différence des arêtes entre sommets "numériques" dont la valeur est fixée au préalable, la longueur des arêtes entre sommets "analogiques" est estimée lors de la recherche de l'arbre d'interconnexions. L'estimation de coût d'arête est présentée dans dans la section ultérieure 4.4.3.

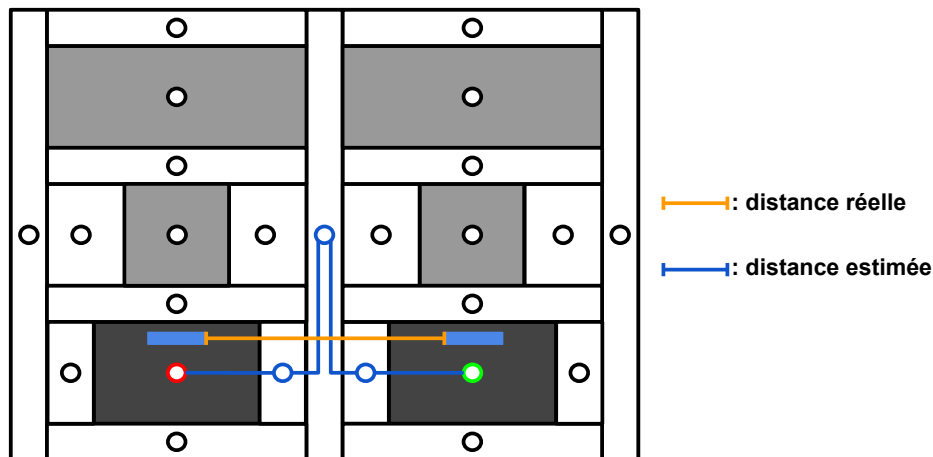


FIGURE 4.12 – Estimation de la longueur de fils d'un *net* d'un circuit analogique en utilisant les distances centre à centre

Nombre de VIAs

Comme mentionné dans la partie 4.2.2, la minimisation du nombre de *VIAs* permet de minimiser la dégradation des performances du circuit et des ressources de routage tant pour les circuits numériques qu'analogiques. Pour cela, il est usuel d'introduire un coût pour l'ajout d'un *VIA* lors de la recherche de l'arbre d'interconnexions (voir 4.2.2). La gestion de ce coût est importante car un coût trop important risque de générer des zones sur-congestionnées et un coût trop faible entraîne des chemins en escalier comprenant un grand nombre de *VIAs*.

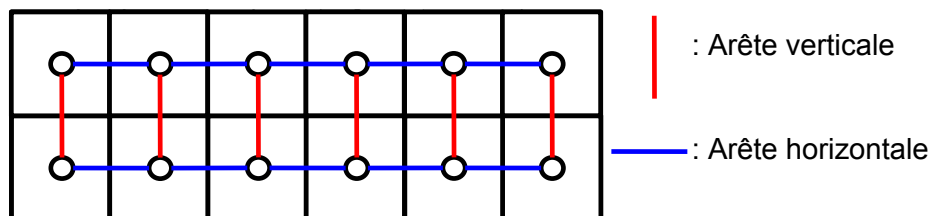


FIGURE 4.13 – Arêtes de type *horizontal* en bleu et *vertical* en rouge

Pour la partie du graphe de routage représentant le circuit numérique, chacune des arêtes connectant deux sommets de pavés numériques est marquée d'un type *horizontal* ou *vertical* (voir figure 4.13). On définit comme arête de type *horizontal* les arêtes connectant deux sommets ayant la même ordonnée et réciproquement sont définies comme arête de type *vertical*, les arêtes connectant deux sommets ayant la même abscisse. La création d'un *VIA* revient à déterminer les changements de type d'arêtes lors du parcours du graphe à pavage régulier. De manière générale, le coût d'un *VIA* pour un routeur global numérique équivaut à trois fois le coût en longueur d'une arête numérique.

Pour la partie du routage représentant le circuit analogique, il est nécessaire d'utiliser une approche différente. Contrairement aux arêtes entre sommets numériques, le coût en longueur est très hétérogène. Cela est en partie dû à la topologie du pavage analogique. Considérer un coût de *VIA* qui serait un multiple du coût en longueur entraînerait l'ap-

partition d'un plus grand nombre de fils courts, les fils longs nécessitant l'ajout d'un *VIA* coûteraient plus cher que les fils courts.

Congestion

Dans le cadre de la prise en compte de la congestion pour le graphe numérique, on considère un coût de congestion dépendant de l'occupation de l'arête. De manière usuelle, le coût de congestion est nul tant que le taux d'occupation de l'arête est inférieur à 80% d'occupation. Lorsque le taux d'occupation est inférieur à 80%, le coût de congestion augmente progressivement jusqu'à atteindre une valeur maximale de 10 fois le coût d'une longueur d'arête reliant deux sommets numériques. La fonction de coût de la congestion se présente sous la forme suivante :

$$\text{coutCongestion}(a) = \frac{h}{1 + e^{-k \times (\text{tauxCongestion}(a) - 1)}} \quad (4.3)$$

avec h et k étant des coefficients fixes permettant de paramétrer la valeur maximum du coût et la pente de la courbe. Les valeurs de h et k sont paramétrées à travers des essais expérimentaux sur différents circuits de test pour une technologie donnée (voir la thèse de Damien Dupuis[13]).

La congestion est gérée différemment pour les parties analogiques du circuit. Il n'existe pas de coût de congestion à proprement dit car les espaces de routage sont dimensionnés en fonction du nombre de fils passant par un espace de routage donné. Comme mentionné dans la partie 3.4.3, les coupures du *slicing tree* définissent les espaces de routage, on nomme ces espaces de routage "canaux de routage". La figure 4.14 montre les espaces de routage déduits du placement et ils se divisent en trois catégories : canal de routage de type *vertical*, canal de routage de type *horizontal* et espace de routage fixe de type *strut*.

Les canaux de routage de type *vertical* issus de découpes verticales sont extensibles en largeur. Dans l'exemple de la figure 4.14, les modules numérotés de 1 à quatre et de 6 à 8 sont des canaux de routage de type *vertical*. Réciproquement, les canaux de routage *horizontal*, issus de découpes horizontales sont extensibles en hauteur. Initialement, la largeur des canaux *vertical* et la hauteur des canaux *horizontal* sont fixées à une valeur permettant de placer une piste de routage. Une fois le routage global effectué, une estimation du nombre de fils passant par chacun des espaces de routage sera déterminée. Les tailles de ces canaux de routage seront alors agrandies en conséquence pour permettre de placer l'ensemble des fils y passant. De cette manière, ces espaces de routage ne peuvent être sur-congestionnés.

Certains espaces de routage qu'on nomme *strut* sont issus de la différence de dimensions entre deux nœuds de même niveau hiérarchique du *slicing tree*, dont un des deux est une feuille de ce *slicing tree*. Dans l'exemple de la figure 4.14, le module 5 a une largeur inférieure aux nœuds hiérarchiques *V1* (vert) et *V2* (rouge) créant ainsi des espaces de routage *strut*. Les espaces de routage de type *strut* ne sont pas extensibles et leur congestion est gérée de telle sorte à ce que lorsqu'une arête atteint son occupation maximale, il n'est plus possible de considérer de chemin passant par cette arête.

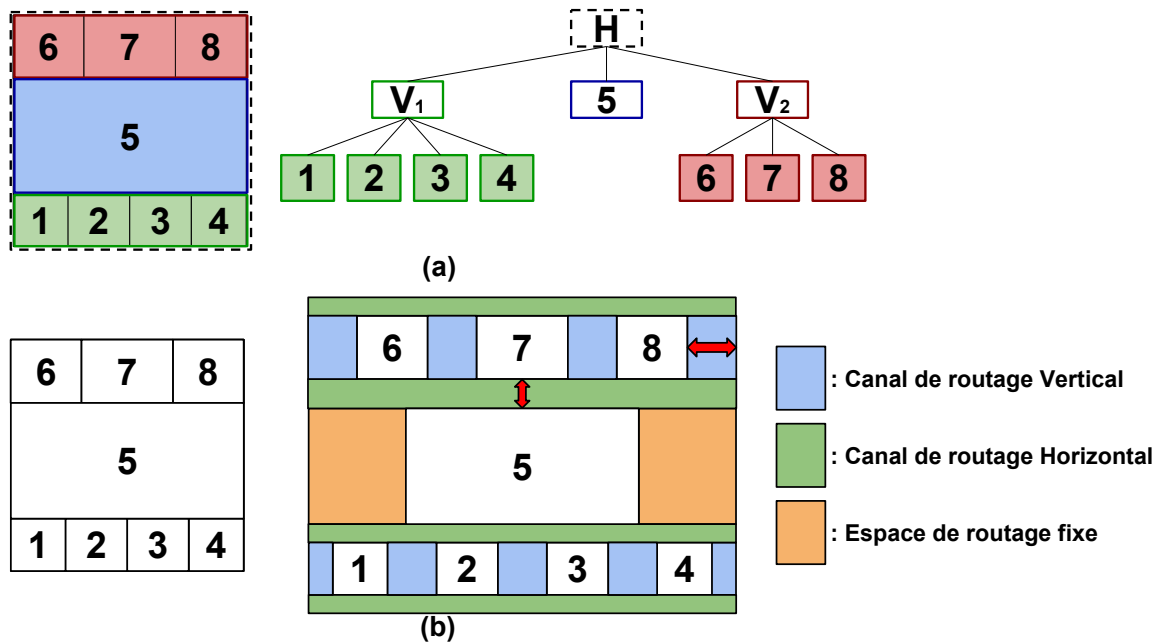


FIGURE 4.14 – *Slicing tree* d'un placement analogique (a) et la représentation de ses canaux de routage (b)

4.4.2 Construction d'arbres d'interconnexions en utilisant l'algorithme de Dijkstra

Les arbres d'interconnexions des *nets* du circuit sont construits en utilisant l'algorithme de Dijkstra. Cet algorithme s'applique sur les parties numériques et analogiques du graphe de routage. La méthode de construction des arbres d'interconnexions est détaillée dans un premier temps avec le cas de deux composantes à connecter et ensuite avec davantage de composantes.

Algorithme de Dijkstra avec une unique source et une unique destination

L'algorithme de Dijkstra est un algorithme permettant de trouver les chemins de coût minimum entre des sommets quelconques d'un graphe connexe. Il existe différentes variantes de l'algorithme et l'originale consiste à trouver le chemin de coût minimum entre deux sommets. La méthode fixe l'un des deux sommets comme étant une source par laquelle l'arbre se propage jusqu'à atteindre l'autre sommet considéré comme destination.

Soit un graphe connexe $G(S,A)$ composé d'un ensemble de sommets S , interconnectés avec un ensemble d'arêtes A dont chacune des arêtes possède un coût et deux sommets nommés *source* et *destination*. Chaque sommet $u \in S$ mémorise :

- **le coût de la chaîne minimale** depuis le sommet *source*, noté $\text{coût}(u)$
- **une référence vers le sommet précédent**, noté *référence*, par lequel la chaîne minimale a atteint le nœud courant

À l'initialisation, le coût de la chaîne minimale du sommet *source* est fixé à $\text{coût}(\text{source}) = 0$ et le coût des autres sommets à $\text{coût}(u) = +\infty$. Afin de gérer la propagation au sein du graphe de routage, on utilise une file de priorité, notée *queue* et les éléments dans la file sont toujours ordonnés en fonction de leur $\text{coût}(u)$. On note également le $\text{coût}(u \rightarrow v)$

permettant d'aller du sommet courant u vers un sommet voisin v . Il est possible d'effectuer trois opérations avec la file de priorité :

- `queue.add(u)` : insérer un sommet dans la file de priorité
- `queue.pop()` : extraire l'élément ayant le plus petit coût(u)
- `queue.empty()` : vérifier si la file est vide

Le déroulement de l'algorithme de Dijkstra pour construire le chemin de coût minimal entre deux sommets est présenté par l'algorithme 1 :

Entrée: un graphe de routage connexe $G(S,A)$
 un sommet $source \in S$
 un sommet $destination \in S$

Sortie : Le chemin de coût minimal à partir de la $destination$ vers la $source$

$coût(u) = +\infty$ pour chaque $u \in S$ // État initial du graphe

$coût(source) = 0$ // Initialisation du coût minimum de la source

`queue.add(source)` // Initialisation de la file de priorité

```

while (queue.empty() != True) do
  |  $u = queue.pop()$  // Extraction de l'élément à plus petit coût( $u$ )
  | if sommet  $u == destination$  then
  | | break; // Fin de l'algorithme
  | end
  | else
  | | foreach (sommet  $v$  connecté à  $u$  par une arête) do
  | | |  $d = \underbrace{coût[u]}_{\text{coût depuis la source à partir de } u} + \underbrace{coût(u \rightarrow v)}_{\text{coût de l'arête}}$ 
  | | | if ( $d < coût[v]$ ) then
  | | | |  $coût[v] = d$ 
  | | | |  $référence[v] = u$ 
  | | | | queue.add(v)
  | | | end
  | | end
  | end
end

```

Algorithme 1: Algorithme de Dijkstra entre deux sommets quelconques d'un graphe connexe

L'exemple de la figure 4.15, tiré de la thèse de Damien Dupuis[13], illustre le déroulement de l'algorithme de Dijkstra. La figure 4.15(a) montre l'état initial d'un graphe de routage en considérant un sommet $source$ et un sommet $destination$. Les cercles représentent des sommets et peuvent contenir un nombre en rouge signifiant le coût minimal temporaire ou un nombre en bleu signifiant le coût minimal définitif après traitement du sommet. Les flèches indiquent la référence vers le sommet précédent retenu comme étant le chemin de coût minimal.

L'algorithme 1 commence par traiter le premier sommet en évaluant le coût vers ses différents sommets voisins. Suite à son traitement, on obtient le résultat de la figure 4.15(b).

L'algorithme se poursuit avec le traitement des voisins du sommet *source* en commençant par le voisin ayant le plus faible coût. La figure 4.15(c) montre une des étapes intermédiaires de l'algorithme sur laquelle les flèches rouges indiquent le prédécesseur retenu jusqu'à cette étape. Le chemin de coût minimal est établi en remontant les prédécesseurs du sommet *destination* jusqu'à atteindre le sommet *source* et est indiqué par les flèches bleues sur la figure 4.15(d).

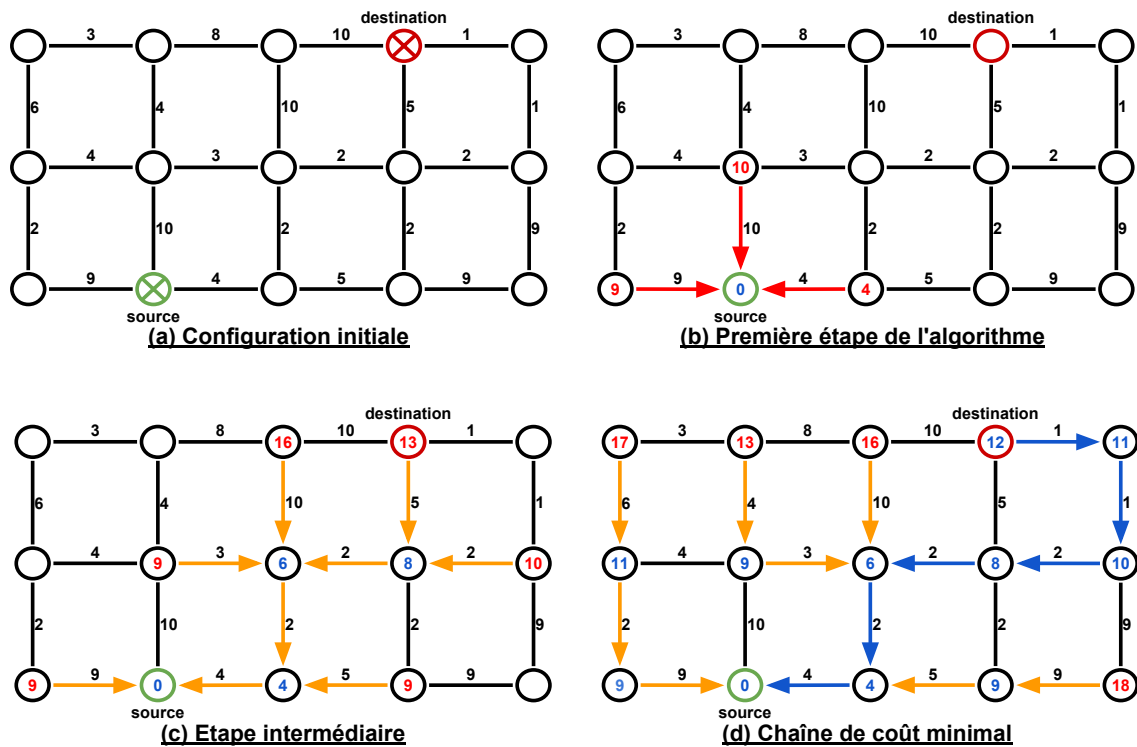


FIGURE 4.15 – Déroulement de l'algorithme de Dijkstra avec deux sommets à connecter

Dans certaines conditions, on est amené à limiter la recherche du chemin de coût minimal à une fenêtre d'exploration. C'est le cas lorsqu'on recherche un arbre d'interconnexions entre sommets représentant des cellules numériques et également lors de recherche d'arbres d'interconnexions pour du routage symétrique (paragraphe 4.2.2). Notons enfin que dans cet exemple simple, la fenêtre d'exploration utilisée est l'ensemble du graphe. Si nous avions utilisé une fenêtre d'exploration égale à la boîte englobante des sommets à relier, la chaîne présentée sur la figure 4.15(c) serait la chaîne de coût minimal.

Gestion de composantes multiples

Le déroulement de la version originale de l'algorithme de Dijkstra s'applique à des situations de bipoints, c'est-à-dire lorsqu'on recherche le chemin de coût minimal entre une source et une destination. Afin de pouvoir l'appliquer à notre problème du routage, il est nécessaire de savoir gérer les situations contenant plus de deux points à connecter. En reprenant l'exemple de la figure 4.15, la figure 4.16.(a) présente un exemple avec trois sommets à connecter.

Le déroulement de l'algorithme démarre de la même façon que dans le cas du bipoint, on détermine un sommet *source* à partir duquel on se propage et on identifie les deux

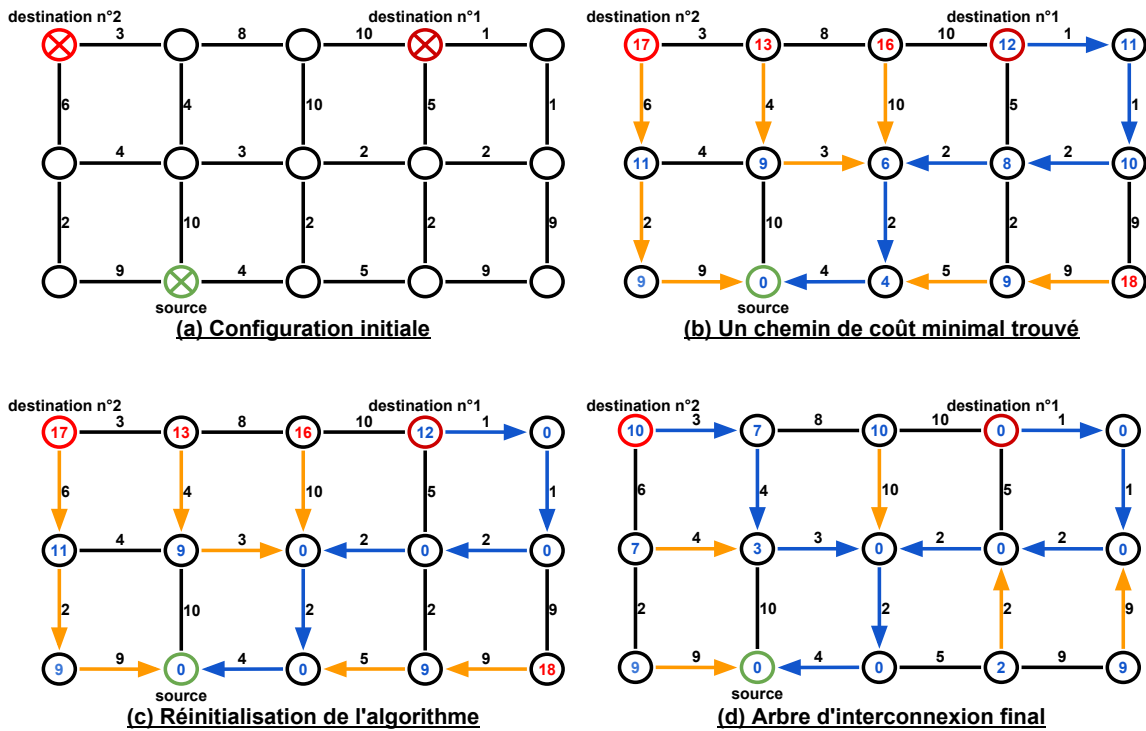


FIGURE 4.16 – Situations avec des sommets ponctuels et non ponctuels

sommets destinations. On reprend l'algorithme 1 avec lequel on détermine un premier sommet *destination* atteint représenté avec la figure 4.16.(b). On réinitialise ensuite la file de priorité avec les deux actions suivantes :

- **Réinitialisation de coûts** : On réinitialise le coût de l'ensemble des sommets compris dans le chemin de coût minimum vers le premier sommet *destination* à la valeur 0.
- **Réinitialisation de la file de priorité** : On insère dans la file de priorité l'ensemble des sommets compris dans le chemin de coût minimum vers le premier sommet *destination*.

Suite à la réalisation de ces deux actions, le graphe devient alors le graphe de la figure 4.16.(c). À partir de ce dernier, on reprend l'algorithme 1 à partir de la boucle **while** qui traitera chacun des sommets insérés dans la réinitialisation de la file de priorité. On note que le coût de chacun des sommets précédemment atteints devient insignifiant. Ces coûts sont redéfinis à partir des nouvelles considérations du sommet *source* qui se voit en quelque sorte étendu. Le résultat du déroulement de l'algorithme de Dijkstra qui permet de trouver le chemin de coût minimal vers le sommet *destination* non atteint est représenté avec la figure 4.16.(d).

Le cas de la figure 4.16 présente un exemple avec trois sommets. On généralise l'algorithme en considérant m le nombre total de sommets au sein d'un graphe connexe $G(S,A)$ composé d'un ensemble de sommets S , interconnectés avec un ensemble d'arêtes A dont chacune des arêtes possède un coût de passage. L'algorithme est présenté par l'algorithme 2.

Entrée: un graphe de routage connexe $G(S,A)$

un sommet $source \in S$

$\forall n \in \{n \in \mathbb{N} | n < (m-1)\}$ sommet(s) $destination \in S$

Sortie : L'arbre d'interconnexions entre le sommet $source$ et les sommets $destination$

$coût(u) = +\infty$ pour chaque $u \in S$ // État initial du graphe

$coût(source) = 0$ // Initialisation du coût minimum de la source

$queue.add(source)$ // Initialisation de la file de priorité

while (Il reste des sommets $destination$ à trouver) **do**

u : Exécution de l'algorithme 1

$w = u$ // Le sommet u est le sommet $destination$ trouvé

 // Phase de réinitialisation des coûts et de la file de priorité

while (($coût[w] \neq 0$) **and** (w a une référence)) **do**

$distance[w] = 0$

$queue.add(w)$

$w = référence[w]$

end

end

Algorithme 2: Algorithme de Dijkstra entre n sommets quelconques d'un graphe connexe

Gestion de composantes connexes

Dans la partie numérique du graphe de routage, le connecteur d'une cellule numérique peut s'étendre sur plusieurs pavages numériques de par sa grande taille. Cela implique qu'il est représenté par plusieurs sommets au niveau du graphe de routage comme indiqué dans l'exemple de la figure 4.17.(a) avec les trois sommets en vert. On qualifie ce cas de source non ponctuelle.

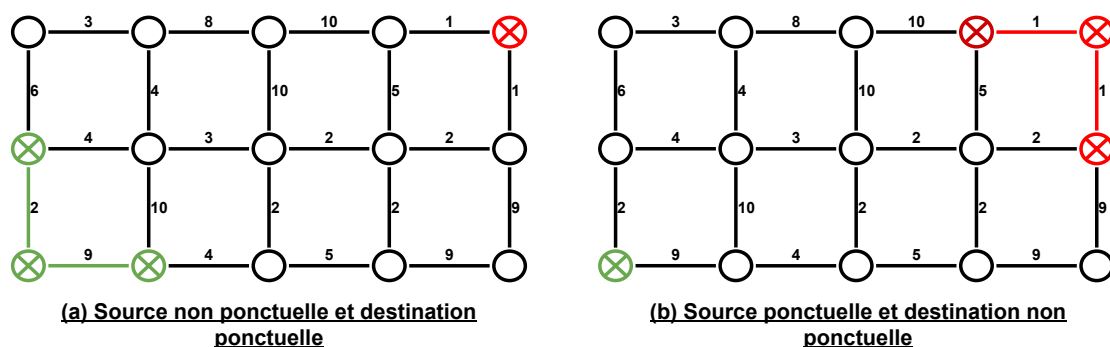


FIGURE 4.17 – Cas de composantes non ponctuelles

L'utilisation de l'algorithme de Dijkstra pour une source non ponctuelle est assez similaire à la gestion de composantes multiples de la partie précédente 4.4.2. L'algorithme de Dijkstra s'initialise en attribuant un coût nul aux sommets $source$ et en les insérant dans la file de priorité.

Il est également nécessaire de considérer la situation dans laquelle le sommet *destination* est une composante non ponctuelle correspondant à l'exemple de la figure 4.17.(b) où le sommet *destination* est représenté par trois sommets. La composante non ponctuelle *destination* est considérée comme atteinte lorsqu'un des sommets le représentant est atteint. En cas de réinitialisation de coût et de file de priorité pour une recherche d'autres sommets, la réinitialisation doit être appliquée à l'ensemble des sommets représentant la composante connexe.

4.4.3 Algorithme de Dijkstra appliqué à la partie analogique du graphe de routage

En plus de la gestion des composantes multiples, notre algorithme de Dijkstra est également adapté pour prendre en compte des contraintes spécifiques aux circuits analogiques. Il existe deux situations particulières liées à la partie analogique du graphe de routage. La première consiste à gérer les situations dans laquelle l'algorithme de Dijkstra se trouve face à des prises de décisions entre des chemins égaux de coût minimum. Sans gestion de ces cas, la solution de l'algorithme peut ne pas être le chemin de coût minimum. L'autre situation à gérer est le cas des *nets* ayant leurs sommets symétriques. La construction de l'arbre d'interconnexions nécessite de trouver un sommet supplémentaire passant par l'axe de symétrie.

Estimation de coût d'arête

L'estimation du coût d'une arête entre deux sommets représentant des modules analogiques est réalisée en calculant la distance Manhattan du chemin optimal pour qu'un fil puisse passer d'un pavé à un autre. Cette distance est calculée à partir d'un point d'entrée du pavé du sommet courant vers un point de sortie vers le pavé du sommet suivant.

On considère l'exemple suivant de la figure 4.18 présentant un placement analogique avec un sommet *source* (en vert) et un sommet *destination* (en rouge) (voir figure 4.18.(a)) et la représentation du graphe de routage correspondant (voir figure 4.18.(b)).

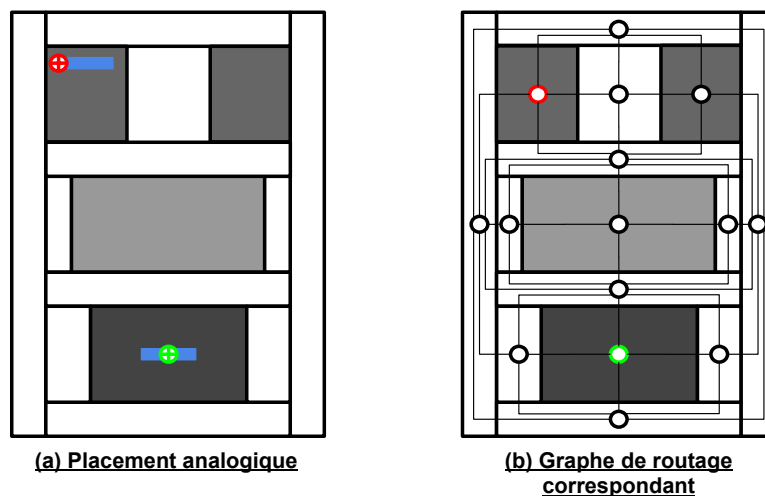


FIGURE 4.18 – Étape 0 - Placement analogique (a) et son graphe de routage correspondant (b)

Pour la partie analogique du routage global, l'estimation du coût d'une arête est calculée en même temps que le déroulement de l'algorithme de Dijkstra. Cette méthode d'es-

timation évalue la distance Manhattan nécessaire pour atteindre la zone représentant le sommet qu'on souhaite atteindre.

La figure 4.19 présente les premières étapes de propagation de l'algorithme de Dijkstra vers le sommet *destination*. Le coût de l'arête du sommet vert vers le sommet bleu équivaut à la distance Manhattan entre le point de démarrage en vert et le point d'entrée en bleu illustré sur la figure figure 4.19.(a). Pour des raisons de clarté, seules les informations pertinentes sont montrées sur le graphe de routage (figure 4.19.(b)).

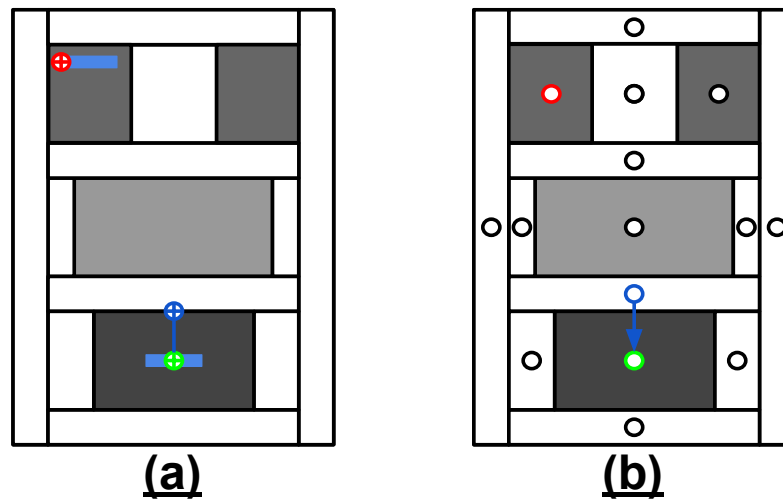


FIGURE 4.19 – Étape 1 - Estimation de coût de chemin (a) et leur graphe de routage correspondant (b)

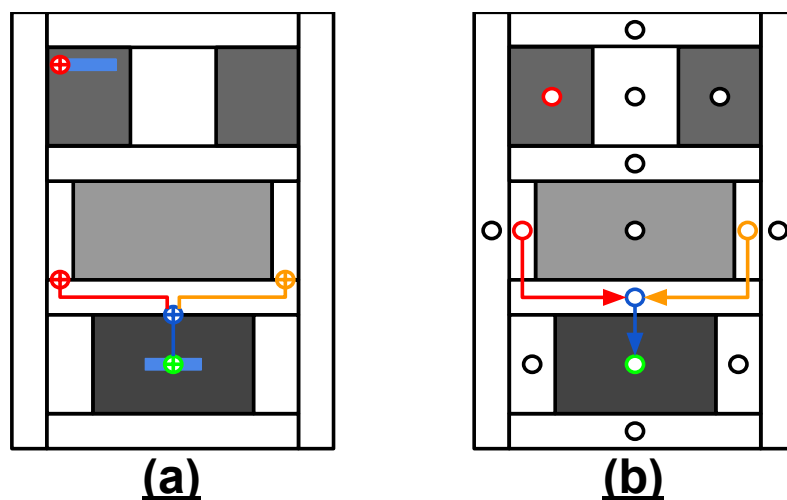


FIGURE 4.20 – Étape 2 - Estimation de coût de chemin (a) et leur graphe de routage correspondant (b)

On déroule une étape supplémentaire de la propagation de l'algorithme de Dijkstra avec la figure 4.20.(a) et 4.20.(b). On considère la propagation de l'algorithme passant par les deux sommets de la figure 4.20(b) en rouge et orange. On remarque que la distance Manhattan vers ces deux arêtes est de même coût. Le bloc central en gris représente un

module et il est impossible de router par-dessus cette zone.

Gestion d'une seconde référence

Certaines situations de propagation de l'algorithme de Dijkstra nécessitent la considération d'une deuxième référence .

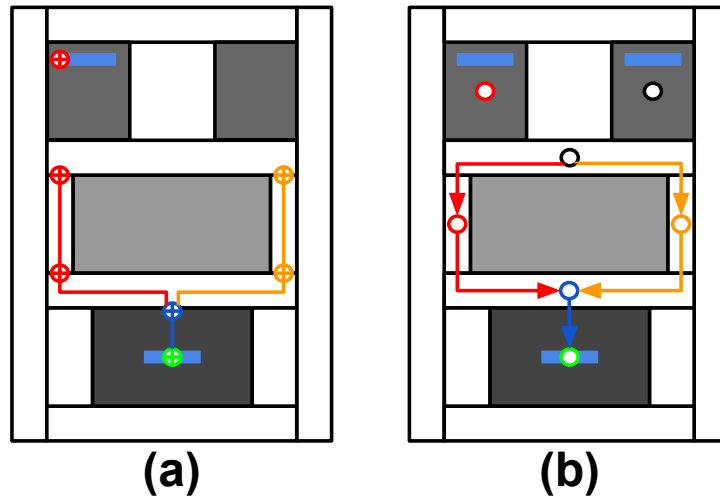


FIGURE 4.21 – Étape 3 - Estimation de coût de chemin (a) et son graphe de routage correspondant (b)

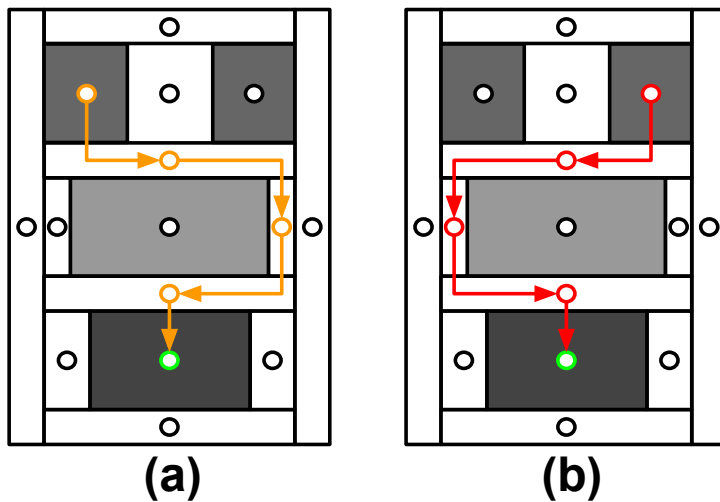


FIGURE 4.22 – Graphe de routage avec erreur d'estimation avec le chemin orange (a) et rouge (b)

La propagation de l'algorithme sur l'exemple précédent nous amène ensuite à la situation de la figure 4.21. On remarque que la propagation du chemin rouge et du chemin orange atteint un canal de routage représenté par un seul sommet. Le coût permettant d'atteindre ce canal de routage est identique entre les deux chemins. Si le sommet du canal de routage ne retenait qu'une référence, le chemin choisi en tant que référence pourrait être un chemin coûtant plus en terme de longueur de fils une fois le signal routé. On se retrouve alors dans une des deux situations de la figure 4.22.

Pour les sommets représentant un module analogique, on considère une deuxième référence vers le sommet précédent lorsque l'algorithme trouve deux chemins égaux de coût minimum. Lorsqu'un sommet possède deux références, le traitement de l'algorithme évalue le coût vers les prochains sommets à partir de ces deux références. La référence par laquelle le chemin est de coût minimum est retenue par le sommet voisin. En reprenant l'exemple précédent, on obtient le résultat de la figure 4.23.

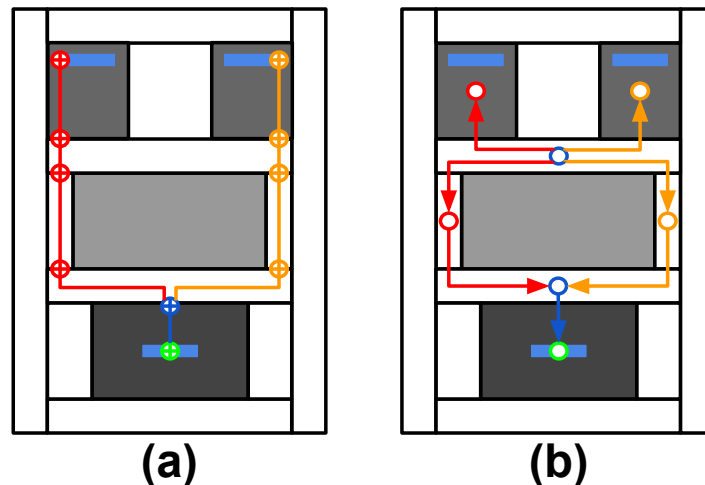


FIGURE 4.23 – Estimation de coût de chemin (a) et son graphe de routage correspondant (b)

Sur la figure 4.23.(b), le sommet bleu ayant les deux références (chemin orange et chemin rouge) évalue le coût vers deux sommets potentiels à connecter (sommet rouge et sommet orange). S'il s'agit du cas du sommet rouge à rejoindre, le coût est minimum en utilisant la référence rouge tandis que si on souhaite rejoindre le sommet orange, le coût minimum en utilisant la référence orange (voir figure 4.23.(a)).

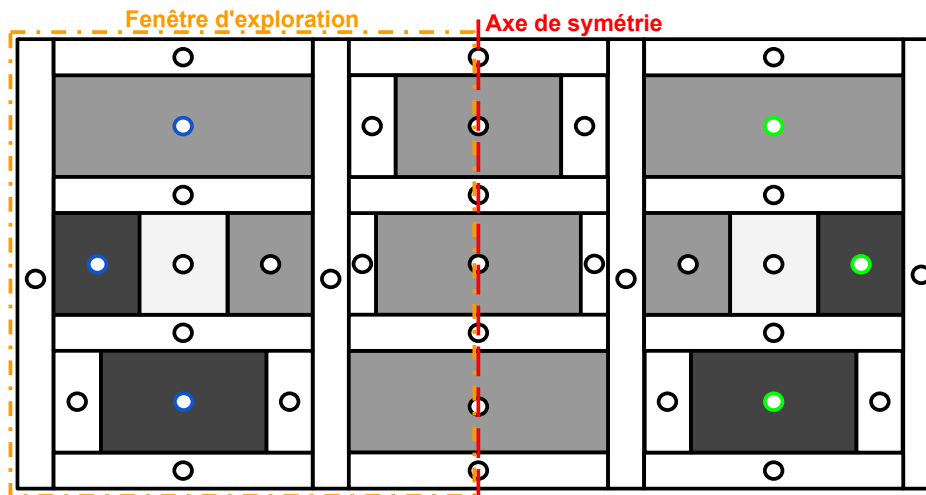
Gestion des constructions d'arbres d'interconnexions avec des symétries

Comme mentionné section 4.2.2, on considère deux cas de symétries dans le cadre du routage global : les symétries entre deux *nets* et les symétries entre les sommets d'un *net* unique.

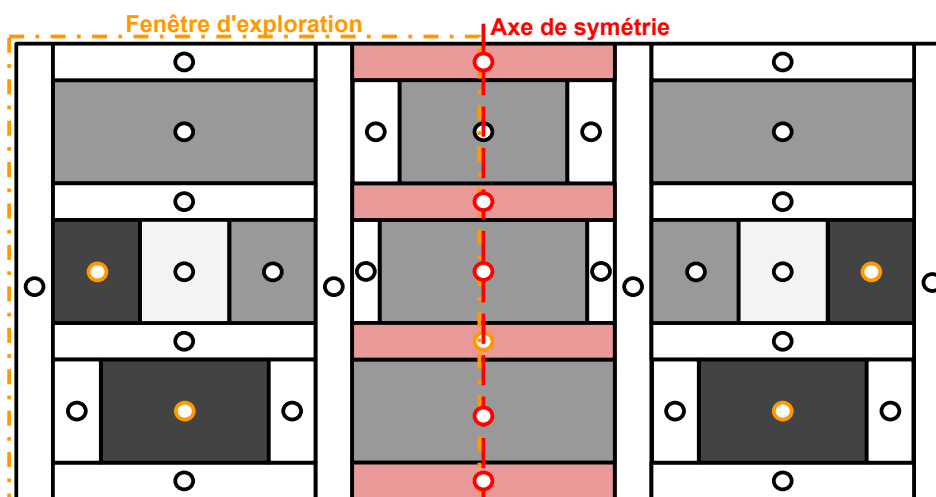
Concernant les symétries entre deux *nets*, cela implique que deux *nets* possèdent le même nombre de sommets et que ces sommets soient placés de façon symétrique par rapport à un axe de symétrie vertical ou horizontal. Cela implique également que les canaux de routage et les modules soient également symétriques de chaque côté de l'axe de symétrie dans une zone comprenant l'ensemble des sommets des deux *nets*.

Le traitement de ce cas de symétrie est réalisé en se limitant à la fenêtre d'exploration comme indiqué en orange sur la figure 4.24. On y observe également les sommets en bleu et en vert composant un *net* symétrique par rapport à l'axe de symétrie en rouge. On utilise l'algorithme de Dijkstra de la partie 4.4.2 dans la fenêtre d'exploration et on réplique l'arbre d'interconnexions obtenu pour les sommets en vert.

Le traitement des symétries entre les sommets d'un *net* unique est similaire avec cependant une considération supplémentaire. De la même façon qu'avec une symétrie entre

FIGURE 4.24 – Symétrie entre deux *nets*

deux *nets*, on limite la fenêtre d'exploration à la moitié des sommets à rejoindre et on utilise l'algorithme de Dijkstra sur la moitié des sommets du *net* comme indiqué en orange sur la figure 4.25.

FIGURE 4.25 – Symétries entre sommets d'un *net* unique

La construction de l'arbre de connexion dans la fenêtre d'exploration doit également inclure la recherche d'un sommet accessible se trouvant au niveau de l'axe de symétrie. Cela implique que l'axe de symétrie passe par ces sommets et que ces sommets ne soient pas un module qui ne fait pas partie du *net* traité. Dans l'exemple, les sommets remplissant ces conditions sont en rouge au niveau de l'axe de symétrie.

L'algorithme de Dijkstra se termine lorsque l'arbre d'interconnexions de coût minimum contenant les sommets de la fenêtre d'exploration et le sommet de l'axe se trouvant le plus proche des sommets de la fenêtre d'exploration est trouvé (voir figure 4.26). Une fois cet arbre d'interconnexions obtenu, on le réplique de l'autre côté de l'axe de symétrie. La combinaison des deux arbres d'interconnexions donne l'arbre d'interconnexions de coût minimum pour le *net* symétrique.

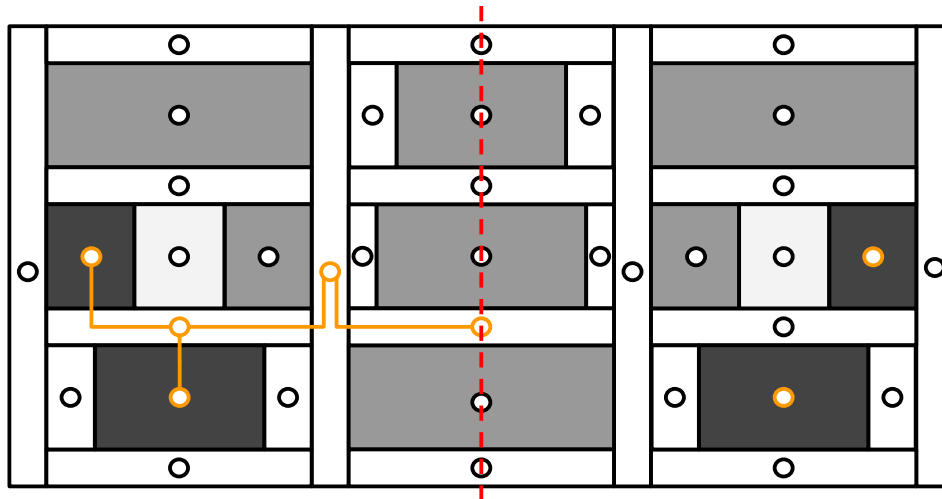


FIGURE 4.26 – Arbre d'interconnexions dans la fenêtre d'exploration

4.5 Implémentation du routage mixte

Dans cette section, nous détaillons l'implémentation du routage global mixte au sein de la plate-forme **Coriolis** dont l'ensemble est majoritairement développé en **C++** et certaines parties en langage **Python**. Notre approche se distingue par plusieurs aspects clés :

- **Structure de données homogène** : Dans l'intégralité de la phase de placement-routage, une structure de données unique est utilisée. Cela permet une concordance des données et une meilleure gestion des données tout au long de la phase de placement-routage. Cette homogénéité de notre structure de données se manifeste lors de la phase de placement et de routage mais également en terme de représentation de composants numériques et analogiques.
- **Graphe de routage mixte** : Comme mentionné dans la section 4.2.1, la grille de routage numérique est régulière tandis que la grille de routage analogique est irrégulière. Cependant, notre outil de routage global manipule un graphe de routage unique dans lequel les graphes de routage numérique et analogique sont connectés en tenant compte des considérations de routage pertinentes pour les deux domaines.
- **Algorithme de construction d'arbres d'interconnexion** : La méthode de construction des arbres d'interconnexions appliquée au sein du graphe mixte est similaire pour les parties numériques et analogiques. Les différences entre le problème du routage numérique et analogique impliquent que l'estimation du coût des arêtes soit différente en fonction de la partie du graphe traité. En numérique, notre outil de routage global estime le coût d'arête rapidement en utilisant la régularité du pavage numérique tandis que l'estimation est plus complexe pour la partie analogique en évaluant plus précisément les coûts d'arêtes.
- **Gestion des contraintes analogiques** : Notre outil de routage global mixte est capable de gérer des contraintes analogiques. La gestion des symétries est réalisée en construisant un demi-arbre d'interconnexions limité par une fenêtre d'exploration à un côté de l'axe de symétrie et en répliquant ce demi-arbre de l'autre côté de l'axe de symétrie. Notre outil est également capable de considérer des fils de différentes

largeurs imposées par le concepteur.

Dans la suite, nous présentons dans un premier temps les différentes étapes constituant la phase de routage global mixte en détaillant ensuite chacune d'entre elles. Nous décrivons l'implémentation de la construction du graphe de routage mixte ainsi que les éléments de notre structure de données permettant de la décrire et de les manipuler lors du déroulement de l'algorithme de Dijkstra. Ensuite, nous expliquons la mise en oeuvre de l'algorithme de Dijkstra avec les détails de l'estimation de coût d'arête et des cas de symétrie. Pour finir, nous présentons la gestion des canaux de routage dans la partie analogique. Pour des raisons de clarté, l'implémentation de la partie numérique est résumée, davantage de détails peuvent être trouvés dans la thèse de Damien Dupuis[13].

4.5.1 Flot du routage global

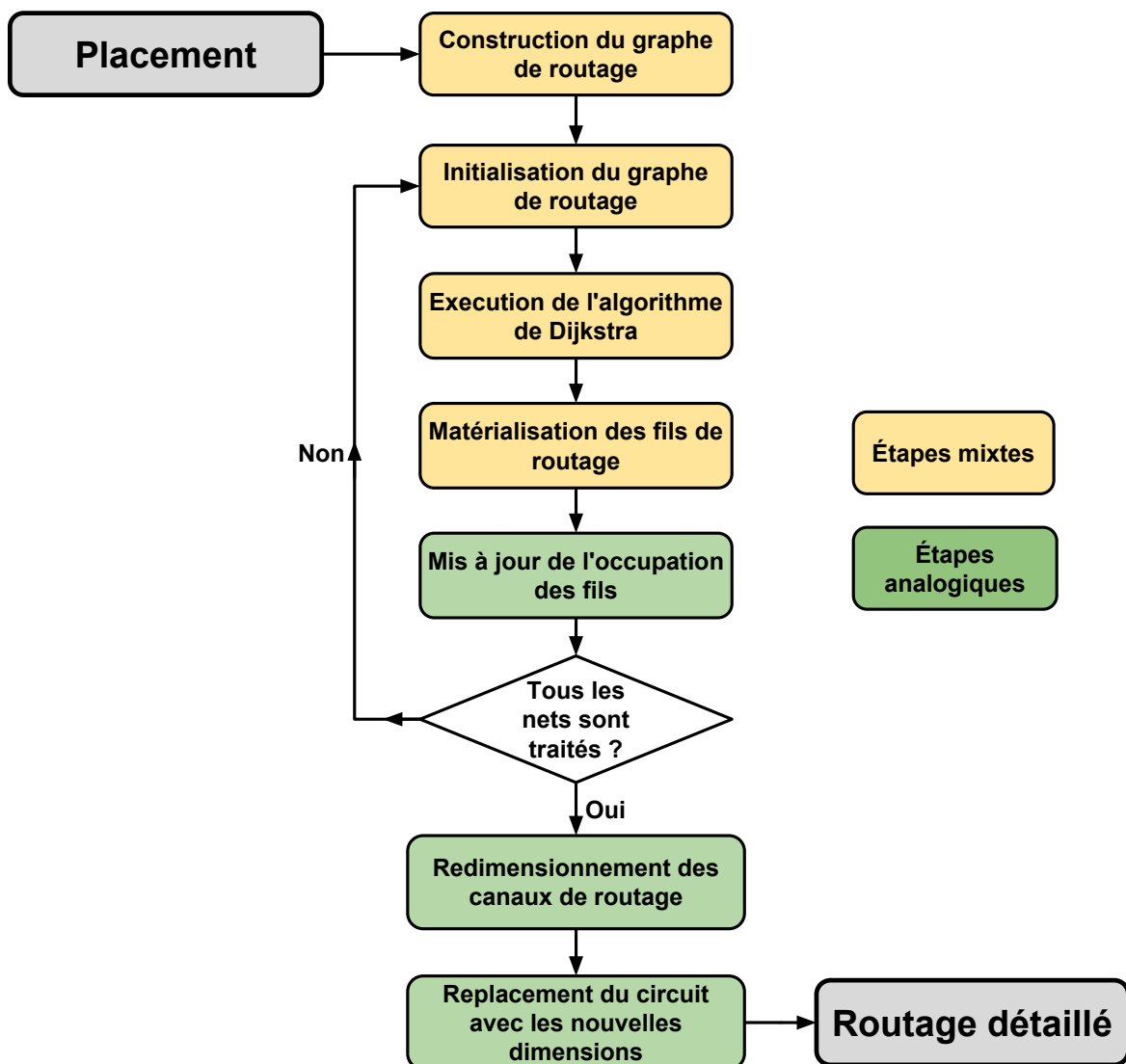


FIGURE 4.27 – Flot du routage global mixte. Les étapes numériques et analogiques du flot sont indiquées en jaune et les étapes concernant uniquement les étapes analogiques sont indiquées en vert

L'implémentation de notre approche suit un ensemble d'étapes représentées par la figure 4.27. Ce flot du routage global est le flot suivi pour router l'ensemble des *nets* du

circuit, cela implique que les *nets* numériques et analogiques sont traités en suivant les mêmes étapes. Les étapes mixtes sont indiquées en jaune et les *nets* analogiques en vert.

Les étapes du routage global mixte sont les suivantes :

1. **Construction du graphe de routage** : Le graphe de routage est créé à partir du résultat de la phase de placement. Chaque arête et sommet du graphe contiennent les informations essentielles telles que leur position, leur dimension, leur référence ou bien leur occupation dans le cas des arêtes. De plus, il existe des fonctions pour associer à chaque position du plan son pavé et son sommet dans le graphe de routage.
2. **Initialisation du graphe de routage** : Cette étape consiste à initialiser les paramètres de chacun des sommets à joindre pour le traitement d'un nouveau *net*. Cette étape définit aussi la fenêtre d'exploration en fonction du cas à traiter comme par exemple la gestion de *nets* exclusivement numériques ou en cas de *net* analogique symétrique.
3. **Exécution de l'algorithme de Dijkstra** : Il s'agit du cœur de la recherche d'arbre d'interconnexions avec le déroulement de l'algorithme de Dijkstra. L'initialisation de la file de priorité et la propagation vers les sommets voisins jusqu'à l'atteindre les sommets cibles y sera détaillée. L'estimation des coûts d'arêtes pour les arbres d'interconnexions analogiques sera également présentée.
4. **Matérialisation des fils de routage** : Une fois les arbres d'interconnexions établis, vient l'étape de création des fils de routage global. Ces fils de routage global sont les données d'entrée de la phase de routage détaillé. Concernant la gestion des *nets* symétriques, on duplique l'arbre d'interconnexions de l'autre côté de l'axe de symétrie en fonction du type de symétrie. La duplication est réalisée en parcourant l'arbre d'interconnexions obtenu suite à l'étape précédente.
5. **Mise à jour de l'occupation des fils de routage** : Une fois l'ensemble des arbres d'interconnexions obtenu, on estime le nombre de fils traversant chacun des canaux de routage de la partie analogique du circuit. Cette opération est réalisée à chaque fois qu'un *net* est traité.
6. **Redimensionnement des canaux de routage** : A la fin du traitement de la totalité des *nets* du circuit, on estime le nombre total des fils passant par chacun des canaux de routage. Les canaux sont redimensionnés de manière à pouvoir contenir tous les fils y passant et ce redimensionnement entraîne un remplacement du circuit.

Les parties suivantes détaillent l'implémentation de chacune de ces étapes.

4.5.2 Construction du graphe de routage global

Pour rappel, le graphe de routage global mixte est un graphe connexe $G(S,A)$ représenté à partir de sommets S , représentant un circuit numérique ou un module analogique, et d'arêtes A , représentant les relations d'un sommet avec ses sommets adjacents. L'implémentation du routage global se doit d'être optimisée en ce qui concerne la structure de données afin de minimiser les ressources en mémoire et garantir un temps d'exécution raisonnable pour la partie numérique du graphe de routage.

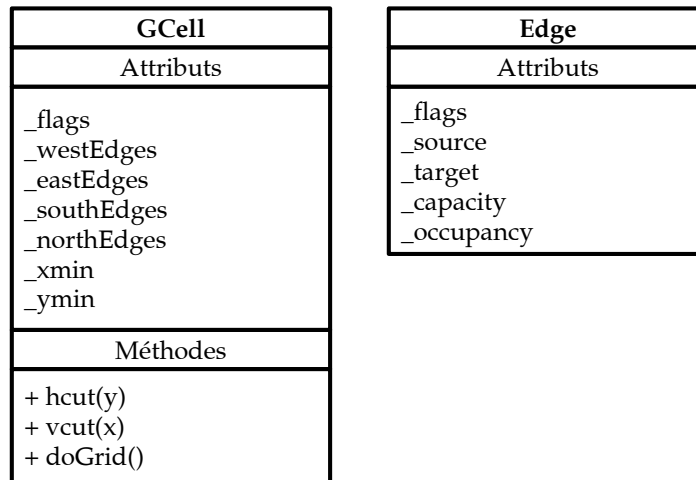


FIGURE 4.28 – Classes utilisées pour la construction du graphe de routage global

L'implémentation du graphe de routage se base essentiellement sur deux classes :

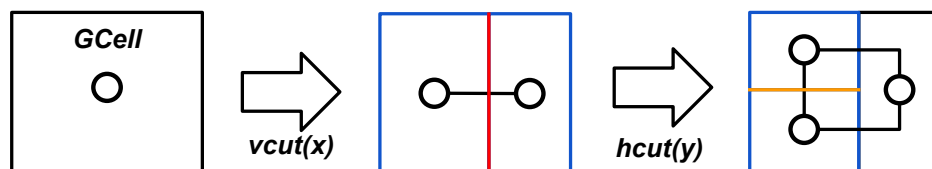
- **class GCell** : Une *GCell* est utilisée pour réaliser le pavage numérique et analogique du circuit. Chaque *GCell* est de forme rectangulaire et contient une arête pour chaque pavé adjacent à elle-même. Ces arêtes sont réparties selon quatre attributs : *_westEdges* comprenant les arêtes vers le côté Ouest du pavé, *_eastEdges* comprenant les arêtes vers le côté Est, *_southEdges* comprenant les arêtes vers le côté Sud et *_northEdges* comprenant les arêtes vers le côté Nord. Cette organisation permet un parcours du graphe plus rapide.

La position d'un pavé est également mémorisée et est décrite par la coordonnée du coin inférieur gauche (*_xmin* et *_ymin*). Les informations de largeur et de hauteur peuvent être calculées à partir de la position des pavés voisins et ne nécessitent donc pas d'être mémorisées. Un attribut *_flags* permet d'enregistrer d'autres paramètres comme par exemple le fait que la *GCell* soit numérique ou analogique.

- **class Edge** : Un *Edge* représente une arête reliant deux *GCells* permettant de connaître les pavages adjacents. Deux attributs *_source* et *_target* représentent les deux sommets reliés par l'arête, la *GCell* *_source* représentera la *GCell* dont la position est la plus basse ou bien la plus à gauche. On y stocke également la capacité (*_capacity*) de piste pouvant passer entre les deux *GCells*, le nombre de pistes occupées (*_occupancy*). Un attribut *_flags* permet de conserver divers paramètres comme par exemple le fait que l'*Edge* soit de type vertical ou horizontal.

Les deux méthodes principales réalisant la construction du graphe de routage sont les méthodes *hcut(y)* et *vcut(x)* qui ont respectivement pour but de diviser le plan de masse en coupant horizontalement et verticalement une *GCell*. À partir d'une *GCell* et d'une coordonnée *y* (resp. *x*), la méthode *hcut(y)* (resp. *vcut(x)*) découpe horizontalement (resp. verticalement) la *GCell* à la coordonnée *y* (resp. *x*) tout en réajustant et en créant les arêtes adéquates avec les *GCells* adjacentes.

La figure 4.29 illustre un exemple d'utilisation des méthodes de découpe. On y observe les *GCells*, marquées en bleu, subissant une découpe verticale (ligne rouge) et une découpe horizontale (ligne orange). Cette découpe entraîne la création d'une nouvelle *GCell* ainsi que la création des arêtes correspondantes avec les pavés adjacents.

FIGURE 4.29 – Exemple d'utilisation des méthodes $vcut(y)$ et $hcut(x)$

La méthode $doGrid()$ est destinée à la création du pavage numérique. De par la régularité du pavage numérique, la méthode de création du graphe de routage numérique est toujours la même pour tout placement donné. $doGrid()$ est une association de l'utilisation des méthodes $hcut(y)$ et $vcut(x)$ et un exemple de transformation en pavage régulier est illustré avec la figure 4.30.

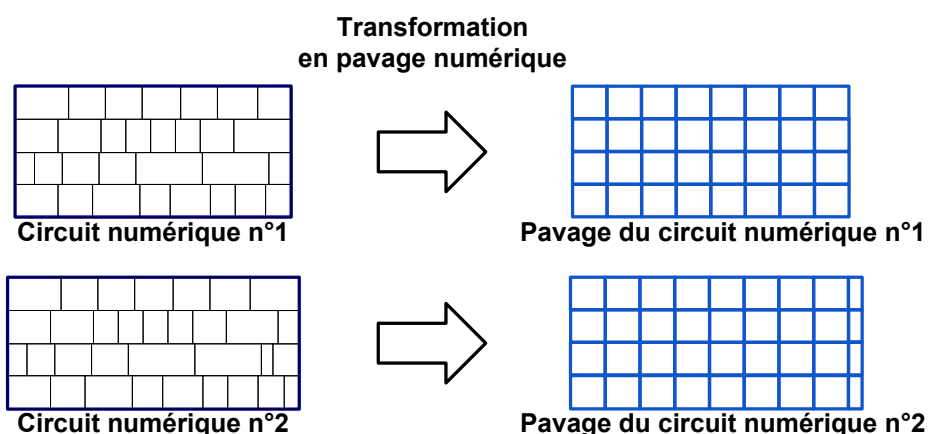


FIGURE 4.30 – Exemple de création de pavage numérique

On observe sur la figure 4.30 que le placement de type numérique ne produit pas toujours une dimension permettant un pavage parfaitement régulier, il est alors possible de réduire la taille des derniers pavés sur un bord de la grille comme dans le cas du second circuit numérique.

Quant aux circuits analogiques, la construction du graphe de routage est réalisée en utilisant les méthodes $hcut(y)$ et $vcut(x)$ en parcourant en profondeur le *slicing tree* du placement. Chaque module et chaque canal de routage du *slicing tree* est représenté par une *GCell*. La figure 4.31 montre l'évolution du parcours d'un *slicing tree* ainsi que les pavages créés. Les *GCells* de type canal de routage n'apparaissent pas sur le graphe.

Un nœud terminal d'un *slicing tree* peut représenter un bloc de modules numériques. Dans ce cas, la *GCell* représentant ce bloc est transformée en pavage numérique comme indiqué sur la figure 4.30.

Afin d'observer le résultat d'un graphe de routage plus complexe, la figure 4.32 présente un circuit virtuel composé de modules numériques, de modules analogiques et de canaux de routage variés. Sur cette figure, seuls les pavés (rectangles) et leurs sommets respectifs (cercles) sont représentés pour des raisons de clarté, chacun des sommets possède une arête vers leurs sommets adjacents. Chaque pavé est mémorisé par son équi-

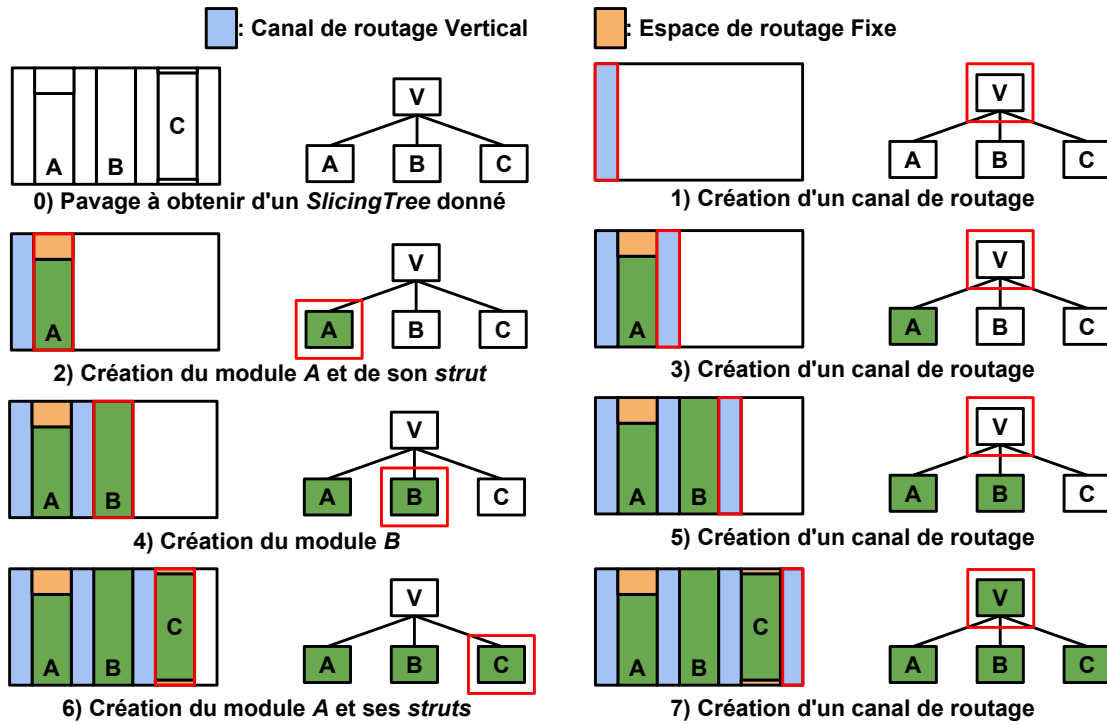


FIGURE 4.31 – Création du pavage pour un nœud vertical d'un *slicing tree*

valence au sein du *SlicingTree*, les pavés liés aux canaux de routage sont mémorisés au niveau de leur nœud hiérarchique et les *struts* sont accessibles par les nœuds *devices* associés.

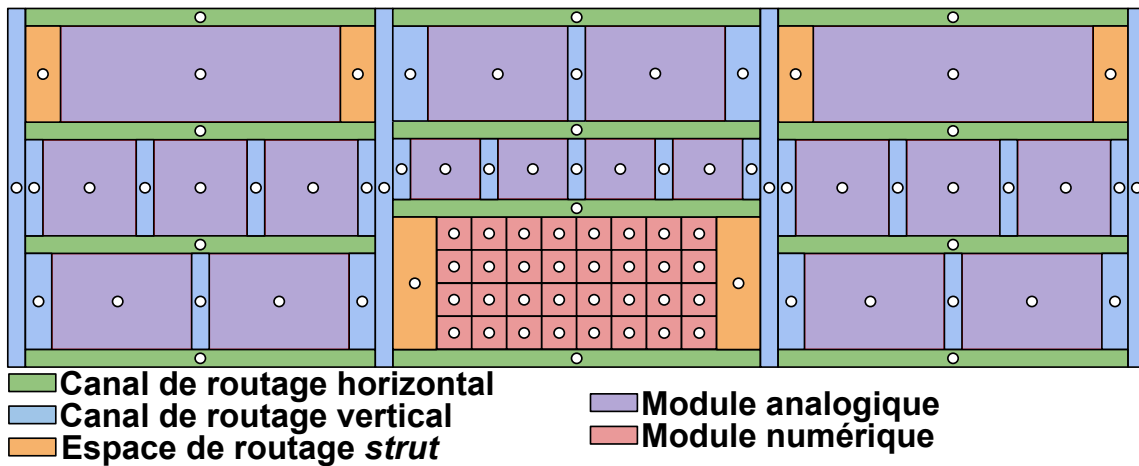


FIGURE 4.32 – Exemple de graphe de routage mixte

Le graphe de routage est prêt à être employé pour la construction des arbres d'interconnexions. Cette construction est réalisée *net* après *net* avec un ordonnancement arbitraire entre les *nets* numériques et analogiques. Le traitement de chaque *net* débute par une étape d'initialisation du graphe de routage. La partie suivante présente l'ensemble des éléments de notre structure de données utilisés pour la mise en œuvre de l'algorithme de Dijkstra.

4.5.3 Initialisation de l'algorithme de Dijkstra

La recherche d'arbres d'interconnexions fondée sur l'algorithme de Dijkstra n'est pas directement réalisée à partir des *GCells* mais avec un ensemble d'éléments plus légers en terme de ressources mémoires. Cela a pour but de minimiser l'utilisation de ressources mémoires pour les circuits numériques volumineux. On utilise les *GCells* dans le but de décrire la représentation du pavage du circuit contenant les informations géométriques et on utilisera la classe *Vertex* pour représenter un sommet du graphe de routage. Elles contiendront les informations nécessaires à la réalisation de l'algorithme de Dijkstra.

Vertex	
Attributs	
	_flags
GCell*	_gcell
Edge*	_from
	_distance
	_stamp
GRADData*	_adata

GRADData	
Attributs	
Interval*	_intervfrom
Interval*	_interv
Interval*	_intervfrom2
Edge*	_from2

Interval	
Attributs	
	_flags
	_min
	_max
	_axis

FIGURE 4.33 – Classes utilisées pour l'algorithme de Dijkstra

On utilise la classe *Vertex* pour l'exécution de l'algorithme de Dijkstra et pour représenter des sommets numériques ou analogiques. On utilise deux éléments en plus pour mémoriser des informations supplémentaires liées aux sommets analogiques : la classe *GRADData* et la classe *Interval* :

- **Vertex** : Un *Vertex* est un sommet du graphe de routage correspondant à un pavage du circuit représenté par l'attribut *_gcell*. Un *Vertex* contient une référence vers le sommet précédent par lequel la chaîne minimale a atteint ce *Vertex*. Cette référence *_from* est l'arête *Edge* reliant ces deux sommets. *_stamp* correspond à un identificateur de *net* (celui du dernier *net* ayant traité le *Vertex*) et permet de savoir si les informations du *Vertex* sont à jour. L'attribut *_distance* indique le coût vers un sommet *source* de l'algorithme de Dijkstra. *_adata* contient des informations en relation avec les *Vertex* analogiques et n'est pas exploité pour les circuits numériques. Un attribut *_flags* permet le marquage de divers paramètres. Il est possible de restreindre l'utilisation des arêtes se propageant par une des faces du *Vertex* (Nord, Sud, Est et Ouest).
- **GRADData** : La classe *GRADData* permet de mémoriser des informations concernant l'occupation des fils dans le *Vertex* courant avec l'attribut *_interv* si ce *Vertex* a été considéré comme étant sur un chemin de coût minimum (voir la partie 4.4.3). En plus de la référence *_from* vers le précédent sommet, on mémorise également l'occupation des fils vers ce sommet lors de la propagation avec l'attribut *_intervfrom*. Comme expliqué dans la partie 4.4.3, *_from2* permet la mémorisation d'une seconde arête vers un deuxième sommet de coût minimum dans certaines conditions qui est accompagné des informations d'occupation de fils *_intervfrom2*. Plus de détails concernant la classe *Intervals* seront fournis lors des explications de la mise en œuvre de l'algorithme de Dijkstra.
- **Interval** : La classe *Interval* est utilisée pour représenter l'occupation des fils dans

une *GCell*. Elle est décrite à partir d'un axe *_axis* considéré vertical ou horizontal et d'un intervalle de valeurs représentant des abscisse ou des ordonnées selon le contexte d'utilisation. Plus de détails concernant la classe *Interval* seront fournis lors des explications de la mise en oeuvre de l'algorithme de Dijkstra.

L'initialisation du graphe de routage correspond à l'initialisation des paramètres des *Vertex* à joindre. L'ensemble des étapes de la phase d'initialisation est illustré par la figure 4.34.

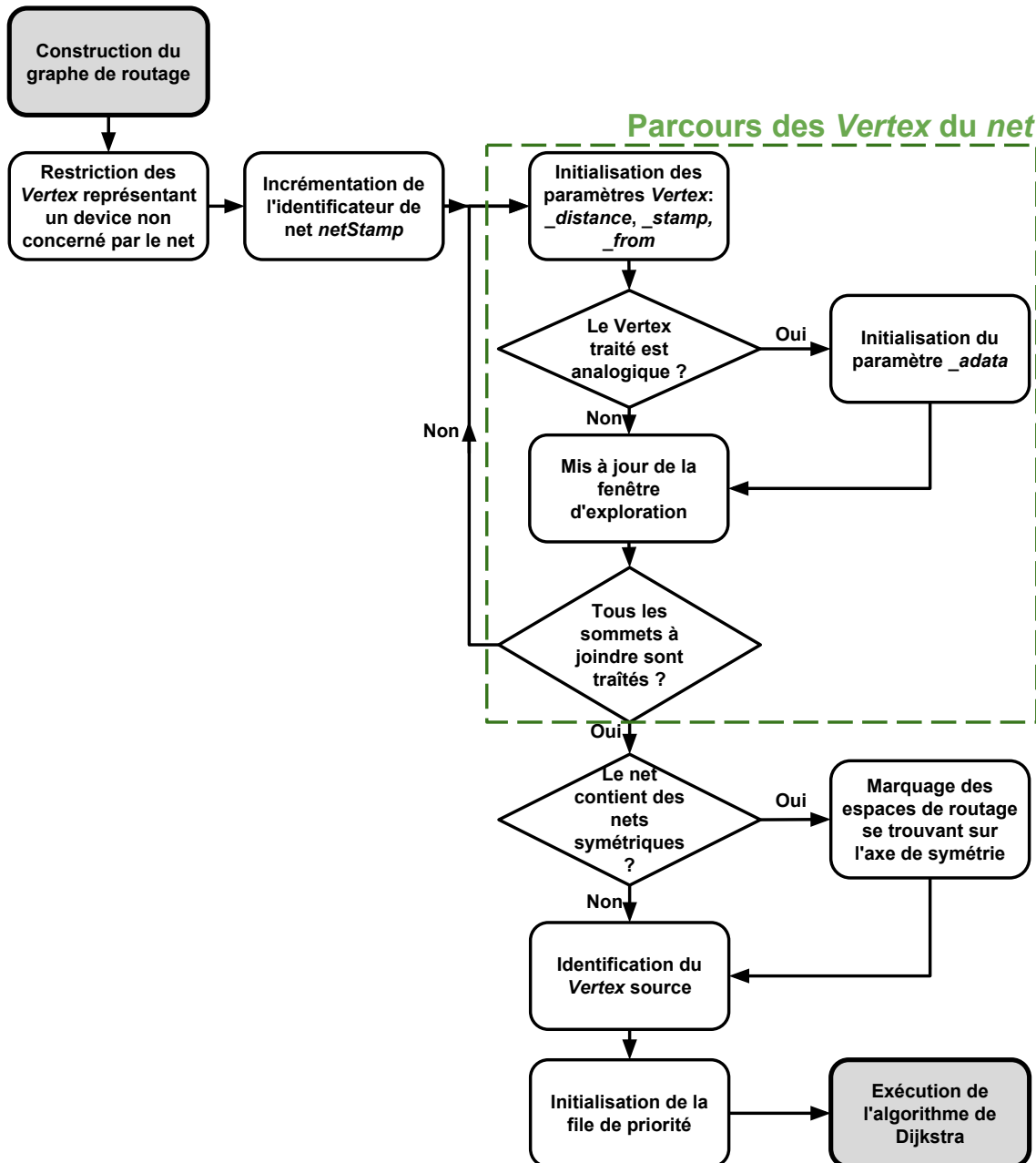


FIGURE 4.34 – Étapes d'initialisation du graphe de routage

La phase d'initialisation de l'algorithme de Dijkstra passe par les étapes suivantes :

1. **Restriction des *Vertex* des modules analogiques** : Afin d'éviter du *routing-over-the-cells*, les *Vertex* des modules analogiques non concernés par le traitement du *net* en cours sont rendus inutilisables lors de la recherche d'arbres d'interconnexions.

2. **Mis à jour de l'identificateur de *net*** : L'identificateur *_netStamp* permet d'identifier le *net* en cours de traitement. Le premier *net* traité est associé à la valeur 0, puis l'identificateur est incrémenté à chaque nouveau *net* traité. La valeur de l'identificateur est comprise entre $\{0, \dots, n-1\}$, où n est le nombre de *nets* à router du circuit. Les attributs *_distance*, *_from* et *_adata* sont pertinents uniquement lorsque le *_stamp* du *Vertex* est identique à l'identificateur *_netStamp*. Dans le cas contraire, *_distance* équivaut à un coût infini, *_from* ne réfère à aucune référence et les informations de *_adata* ne sont pas pertinentes.
3. **Initialisation des *Vertex* à joindre** : La plate-forme **Coriolis** intègre des fonctionnalités d'accès vers les différents composants du circuit. Il est possible d'obtenir les informations concernant les connecteurs faisant partie d'un même *net*, en particulier la position des segments représentant les connecteurs. On utilise cette position avec une fonction précédemment introduite par Damien Dupuis[13] permettant d'accéder à une *GCell* du circuit à partir d'une position du plan. De cette manière, on peut obtenir l'ensemble des *Vertex* représentant un module analogique à router. Cette étape consiste à initialiser les *Vertex* des modules analogiques concernés par le *net* en cours. *_distance* est initialisé à 0, *_stamp* prend la valeur de l'identificateur du *net* courant, la référence vers le prédécesseur *_from* est considéré *NULL*.
4. **Initialisation des informations analogiques complémentaires** : Dans la mesure où un *Vertex* à rejoindre représente un module analogique, l'attribut *_adata* est créé et les informations d'occupation de fils sont initialisées à partir des dimensions du connecteur à rejoindre. Dans le cas d'un *Vertex* représentant une *GCell* numérique, l'attribut *_adata* est considéré *NULL* afin de ne pas surcharger la consommation mémoire dans la gestion des *Vertex* numériques.
5. **Mise à jour de la fenêtre d'exploration** : La fenêtre d'exploration correspond à la zone dans laquelle l'algorithme de Dijkstra est réalisé. Seuls les *Vertex* se trouvant dans la fenêtre d'exploration sont considérés. On distingue quatre situations :
 - Le *net* comporte uniquement des *Vertex* représentant des modules numériques, la fenêtre d'exploration correspond à la zone minimale comprenant l'ensemble des *Vertex* du *net*.
 - Le *net* comporte uniquement des *Vertex* représentant des modules analogiques, la fenêtre d'exploration correspond à la zone minimale comprenant l'ensemble du circuit analogique.
 - Le *net* comporte uniquement des *Vertex* représentant des modules analogiques et est symétrique, la fenêtre d'exploration correspond à la zone délimitée par l'axe de symétrie et une extrémité du circuit analogique complet.
 - Le *net* comporte des *Vertex* numériques et analogiques, la fenêtre d'exploration correspond à la zone minimale comprenant l'ensemble des *Vertex* numérique et l'ensemble du circuit analogique.
6. **Marquage des *Vertex* se trouvant sur un axe de symétrie** : Pour les *nets* comportant des sommets symétriques, on marque l'ensemble des *Vertex* utilisables se trouvant sur l'axe de symétrie. Cela permet de pouvoir construire la moitié de l'arbre d'interconnexion comme mentionné dans la partie 4.4.3.

7. **Identification du Vertex source** : Parmi l'ensemble des Vertex à joindre, on identifie le *Vertex* source à partir duquel on démarre l'algorithme de Dijkstra comme étant le plus au centre. On calcule le point moyen des *Vertex* à rejoindre et le *Vertex* étant le plus proche de ce point est considéré comme étant le *Vertex* source. Les autres *Vertex* sont considérés comme étant des *Vertex* destinations.
8. **Initialisation de la file de priorité** : Cela consiste à insérer le *Vertex* source dans la file de priorité.

4.5.4 Mise en oeuvre de l'algorithme de Dijkstra

Suite à la phase d'initialisation de l'algorithme de Dijkstra, on utilise l'algorithme de Dijkstra afin de rechercher les arbres d'interconnexions optimaux pour chacun des *nets* numériques et analogiques. Cette recherche est réalisée *net* après *net* avec un ordonnancement arbitraire entre les *nets*. On appelle "*Vertex* source" le sommet à partir duquel l'algorithme de Dijkstra se propage et on appelle "*Vertex* destination", les autres sommets du *net* nécessitant d'être joints.

L'ensemble des étapes de l'application de l'algorithme de Dijkstra est illustré par la figure 4.35 et suit l'algorithme détaillé dans la partie 4.4.3 :

1. **Extraction du Vertex de plus petit coût** : Cette étape consiste à obtenir le *Vertex* en tête de la file de priorité, c'est-à-dire ayant le coût le plus faible de la file de priorité. Les *Vertex* dans la file de priorité sont triés par rapport à leur coût vers le *Vertex* source.
2. **Extraction d'une arête du Vertex courant** : Chaque *Vertex* se réfère à une *GCell* comme étant un pavage de la grille de routage. Une *GCell* possède autant d'arêtes qu'elle a de pavés voisins. Ces arêtes sont organisées en quatre directions : les arêtes du côté Nord, Sud, Est et Ouest.
3. **Estimation du coût d'arête** : Comme il a été mentionné dans les parties précédentes, l'estimation du coût d'une arête est évalué selon les types (numérique/analogique) des *Vertex* courant et voisins. L'estimation du coût d'arête est réalisée par trois vertex : le *Vertex* courant, sa référence et le *Vertex* voisin considéré.
4. **Gestion de la mémorisation de la référence** : Une fois le coût total estimé pour atteindre le *Vertex* voisin, ce coût est comparé avec le coût précédemment estimé par un autre chemin si le *Vertex* voisin a déjà été traité. Le résultat de cette comparaison entraîne la mémorisation du nouveau coût d'arête du *Vertex* voisin et d'une référence vers le *Vertex* courant.
5. **Réinitialisation du graphe de routage** : Cette réinitialisation suit le déroulement détaillé dans la partie 4.4.2 et consiste à la réinitialisation du *Vertex* du chemin d'interconnexions trouvé. Une estimation du passage optimal des fils passant dans chacun des *Vertex* du chemin d'interconnexion est également mémorisé en utilisant la classe *Interval*.

L'algorithme est terminé lorsque l'ensemble des *Vertex* à rejoindre ont été atteints et la condition "*Tous les Vertex destination sont atteints ?*" est remplie. Dans le cas d'un *net*

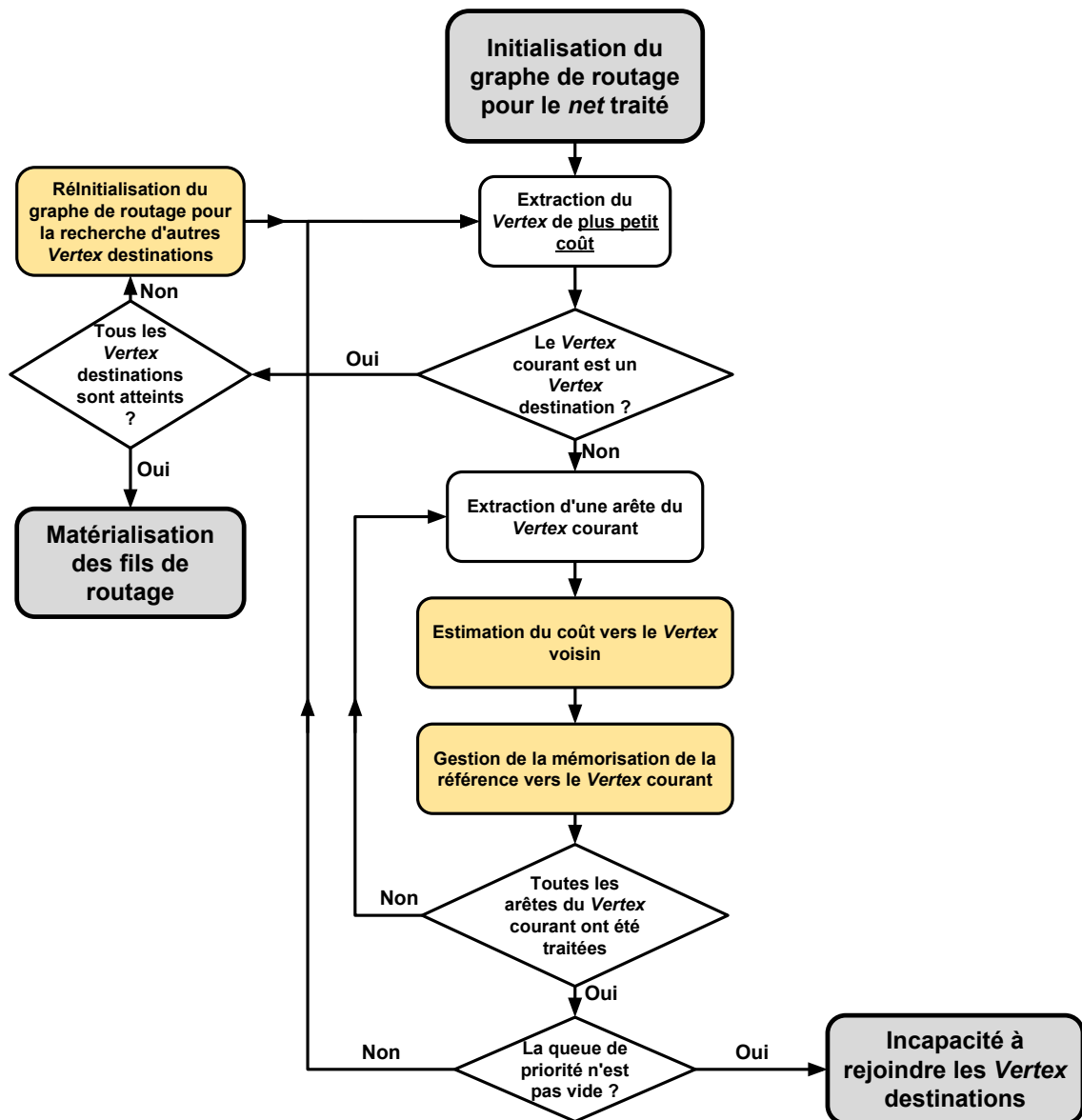


FIGURE 4.35 – Étapes du déroulement de la recherche d'arbres d'interconnexions

dont ses Vertex sont symétriques, il faut atteindre en plus un des Vertex sur l'axe de symétrie (voir la partie 4.4.3).

Parmi ces étapes, "l'estimation du coût vers le Vertex voisin", la gestion de la mémorisation de la référence" et "la réinitialisation du graphe de routage" nécessitent une attention particulière (en jaune sur la figure 4.35). En fonction du type du Vertex courant et du type du Vertex suivant, leur fonctionnement est adapté à la méthode d'estimation de coût approprié ainsi que la gestion au niveau des frontières entre le numérique et l'analogique. Le pavage du circuit étant mixte, on se doit de considérer la gestion de l'estimation du coût lorsque le Vertex courant est numérique (resp. analogique) et le Vertex voisin analogique (resp. numérique). On décrit dans les sous-parties suivantes le fonctionnement de l'estimation du coût d'arête, de la gestion de la mémorisation de la référence et de la réinitialisation du graphe de routage.

Implémentation de l'estimation du coût d'arête

La réalisation de l'estimation du coût d'arête est différente en fonction du type du *Vertex* courant et du *Vertex* voisin. Le routeur global possède deux modes d'estimation de coût d'arête : une estimation numérique et une estimation analogique. L'ensemble des étapes de l'estimation du coût d'arête est illustré par la figure 4.36 :

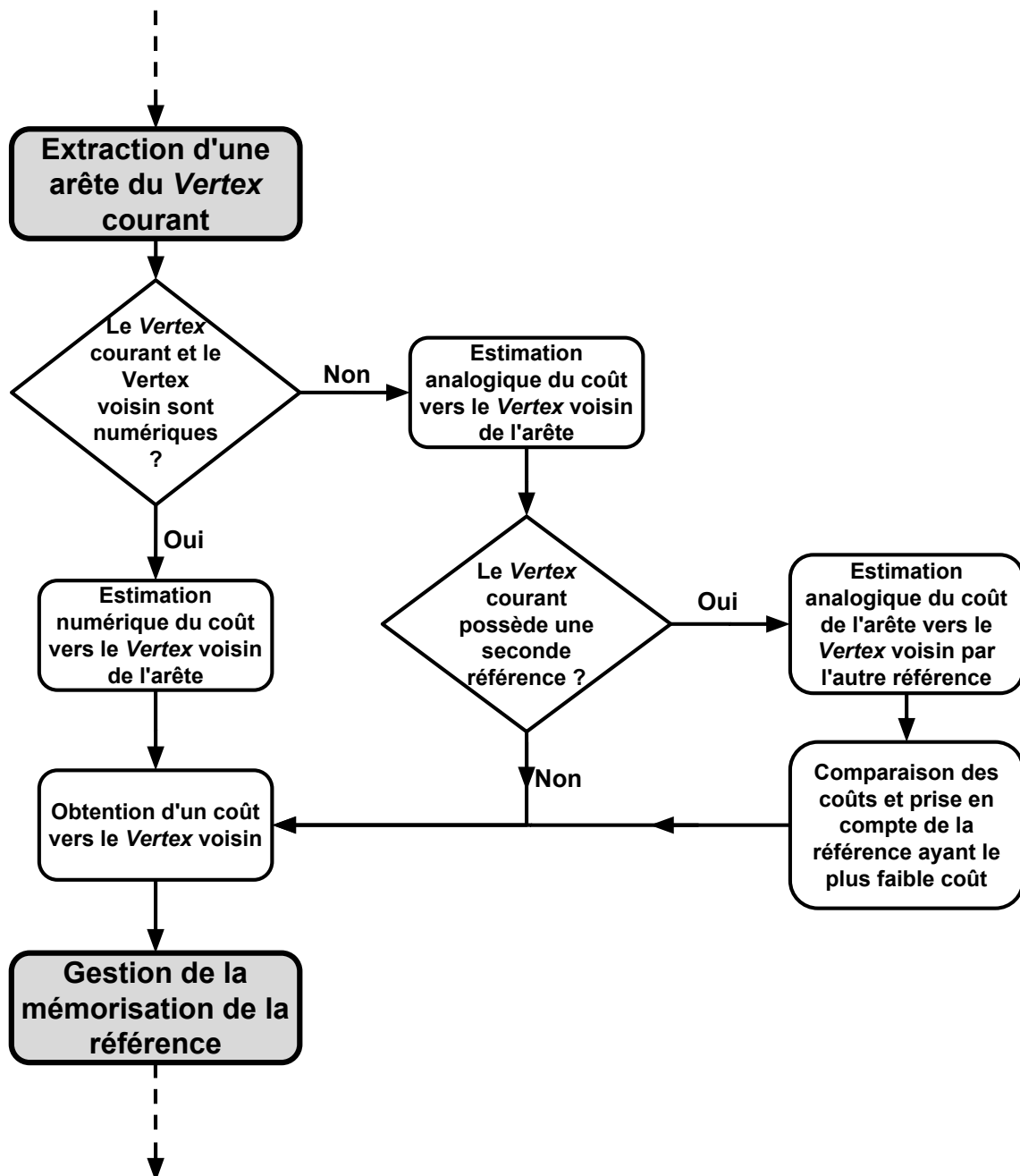


FIGURE 4.36 – Étapes de l'estimation du coût de l'arête en cours

- **Type du *Vertex courant* et du *Vertex voisin*** : Lorsque ces deux *Vertex* sont numériques, le routeur global emploie l'estimation de coût d'arête numérique. Dans le cas contraire, on utilise une estimation de coût d'arête analogique.
- **Estimation du coût d'arête analogique** : L'estimation du coût d'arête analogique est plus précise que l'estimation numérique et se base sur la longueur de fil optimale

passant par le chemin de *Vertex*.

- **Estimation du coût d'arête analogique par une seconde référence** : Pour les raisons mentionnées dans la partie 4.4.3, certains *Vertex* analogiques peuvent contenir une seconde référence. Si c'est le cas, on estime aussi le coût d'arête en utilisant cette seconde référence.
- **Comparaison de coût d'arête** : On considère uniquement la référence par laquelle le coût d'arête est le plus faible pour la suite de la recherche d'arbres d'interconnexions.
- **Obtention d'un coût vers le *Vertex* voisin** : Le coût total pour atteindre le *Vertex* voisin à partir du *Vertex* courant correspond à la somme du coût pour atteindre le *Vertex* courant et du coût d'arête vers le *Vertex* voisin.

Lors de la propagation des arbres dans la recherche d'arbres d'interconnexions, on distingue quatre situations différentes d'estimation du coût d'arête :

- Le *Vertex* courant et le *Vertex* voisin sont numériques
- Le *Vertex* courant et le *Vertex* voisin sont analogiques
- Le *Vertex* courant est numérique et le *Vertex* voisin est analogique
- Le *Vertex* courant est analogique et le *Vertex* voisin est numérique

On rappelle que l'estimation du coût d'arête entre deux sommets numériques adjacents équivaut à la somme du coût de longueur, de *VIA* et de congestion. Le coût de longueur entre deux sommets numériques correspond à la distance centre à centre en distance Manhattan. Le coût de *VIA* correspond à trois fois le coût de la distance et est appliqué lors d'un changement de direction de propagation en passant d'une arête verticale (resp. horizontale) suivi d'une arête horizontale(resp. verticale). Le coût de congestion est évalué avec l'équation de la section 4.3 pour laquelle les paramètres h et k ont pour valeurs respectivement 10 et 30. Ces valeurs sont ajustées suite à des expérimentations sur des circuits de test pour une technologie donnée.

Dans le cas où le *Vertex* courant et le *Vertex* voisin sont analogiques, l'estimation du coût en longueur est le critère le plus critique et nécessite une attention particulière. On rappelle qu'on interdit le *routing-over-the-cell* au dessus des *GCells* représentant un module analogique et que les fils passent uniquement dans les canaux de routage. Afin de comprendre la méthode d'estimation de longueur de fils entre *Vertex* analogiques, considérons la figure 4.37 avec le passage d'un fil dans un canal de routage horizontal.

Dans cet exemple, on souhaite illustrer l'occupation d'un fil passant dans la *GCell* représentant un canal de routage. On considère que ce *Vertex* courant possède une référence vers un *Vertex* se trouvant au Sud (en vert) et par conséquent que le fil arrive par le côté Sud (en vert). Les propagations vers les *Vertex* voisins dans les directions Nord(N), Sud(S), Est(E) et Ouest(O) sont représentées en jaune. La figure 4.37 présente toutes les configurations possibles de passage de fils vers toutes les directions.

Le coût en longueur vers le *Vertex* voisin correspond à la distance Manhattan entre le point entrant par le Sud et le point sortant vers le *Vertex* voisin (voir figure 4.37). À l'exception des *Vertex* source et destination, le point entrant vers le *Vertex* courant est déterminé à partir du point sortant de sa référence. Le point sortant vers le *Vertex* voisin est déterminé à partir de la position du point entrant, le point sortant est situé sur la frontière

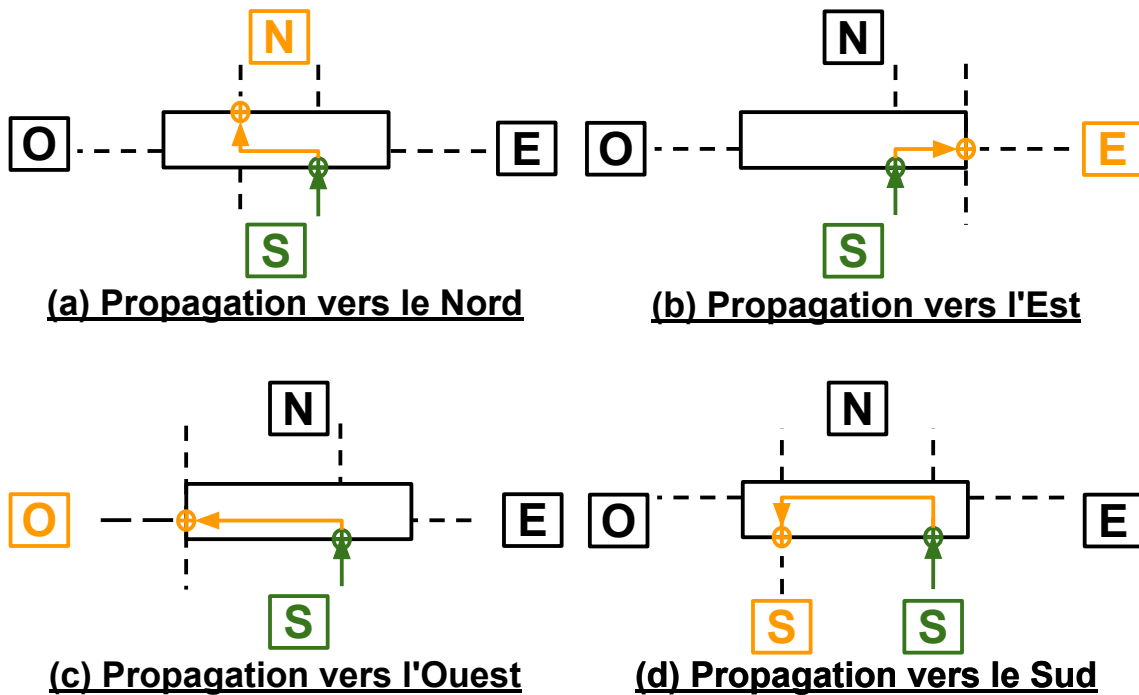


FIGURE 4.37 – Description de l’occupation d’un fil et de son coût en longueur dans un canal de routage horizontal

Vertex courant-*Vertex* voisin et au plus près possible du point entrant.

Pour les *Vertex* source, le point entrant est considéré à partir du connecteur du module analogique. Lors de l’étape d’initialisation des informations analogiques complémentaires, les informations de dimensions du segment connecteur sont stockées par l’attribut *_adata* (voir la section 4.5.3). Connaissant ces informations, on peut en déduire le point entrant du connecteur. Le point sortant vers le *Vertex* est toujours déterminé à la position optimale à partir de la position du point entrant. La figure 4.38.(a) illustre la propagation d’un *Vertex* source vers les différentes directions, le rectangle bleu se trouvant dans la zone grise représente le connecteur/segment à joindre.

Concernant les *Vertex* destination, le point sortant est considéré différemment lorsqu’ils jouent le rôle du *Vertex* voisin. La position du point sortant n’est plus considérée au niveau de la frontière *Vertex* courant-*Vertex* voisin mais à la frontière du connecteur. L’atteinte d’un *Vertex* destination signifie la fin d’une propagation de chemin d’interconnexions, cela implique que la situation de recherche de point entrant pour une propagation n’existe pas. La figure 4.38.(b) illustre la propagation vers un *Vertex* destination en provenance des différentes directions.

En plus du coût en longueur le coût d’arête comprend également le coût de *VIA*. La nécessité d’un *VIA* est déterminée par la position du point entrant et du point sortant. Les plages de variation de la position du point entrant et du point sortant sont comparées. Observons le cas présenté sur la figure 4.39 sur lequel le canal de routage horizontal central (en vert) est le *Vertex* courant :

- Lorsque la propagation s’effectue dans la même direction, l’union des plages de variation de la position du point entrant et du point sortant doit être non nulle. Dans

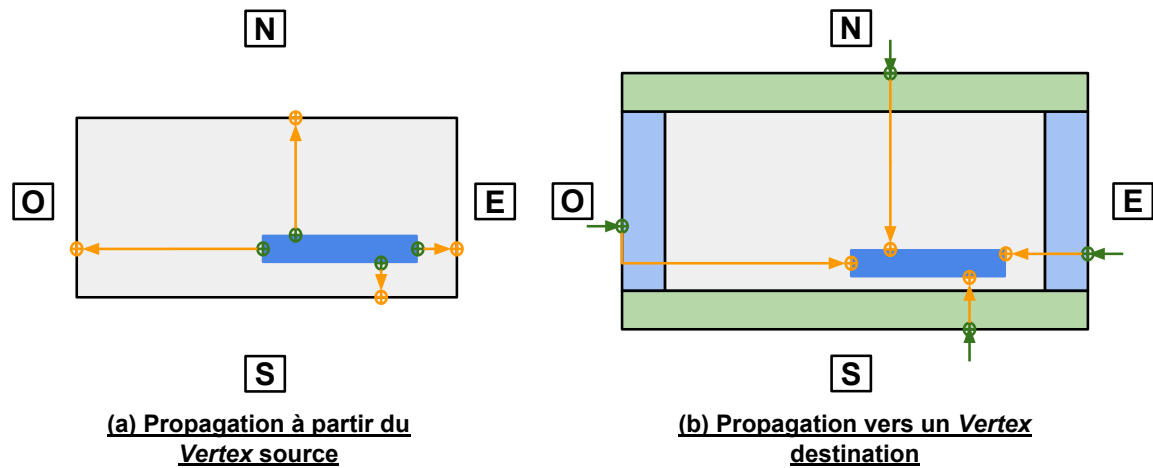


FIGURE 4.38 – Gestion d'estimation de coût de longueur avec un *Vertex* destination et un *Vertex* source

le cas contraire, le fil nécessitera l'utilisation de deux *VIAs* et ajoute donc deux coûts de *VIAs* au coût de l'arête. Sur la figure 4.39.(a), on observe qu'une propagation vers le Nord implique un double coût de *VIAs* pour les *Vertex* se trouvant aux extrémités et un coût de *VIAs* nul pour le *Vertex* voisin central. Sur la figure 4.39.(c), un point entrant provenant du Sud pour une propagation vers un point sortant également vers le Sud entraîne automatiquement un double coût.

- Lorsque la propagation est perpendiculaire, le fil nécessitera obligatoirement l'utilisation d'un *VIA* ajoutant ainsi un coût de *VIA* au coût d'arête. Sur la figure 4.39.(b), la propagation vers l'Est et l'Ouest entraîne l'ajout d'un coût de *VIA*.

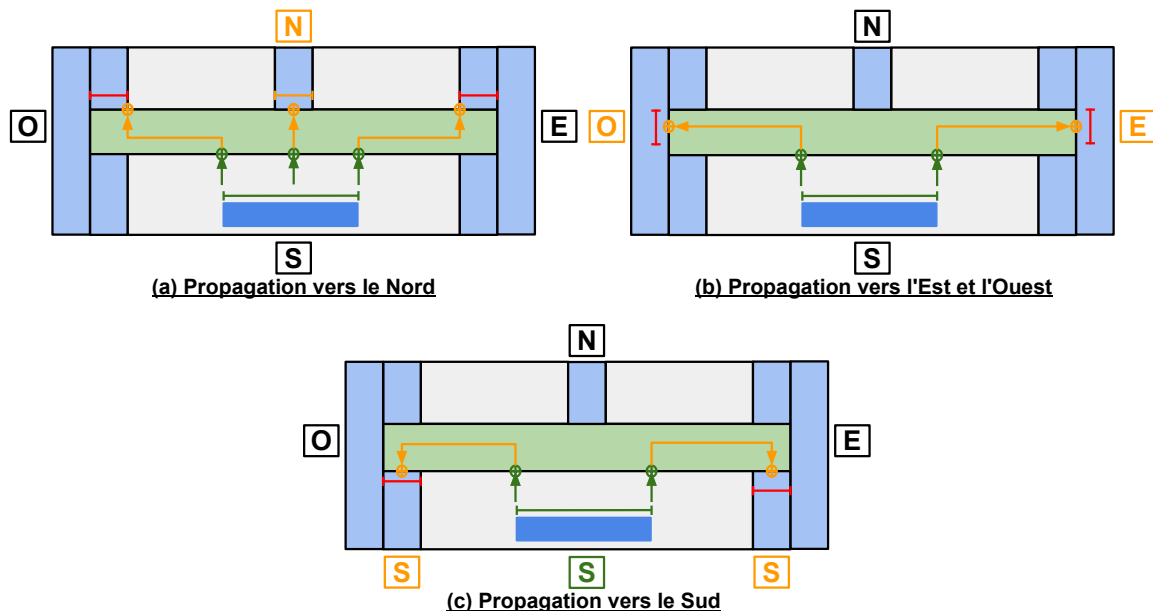
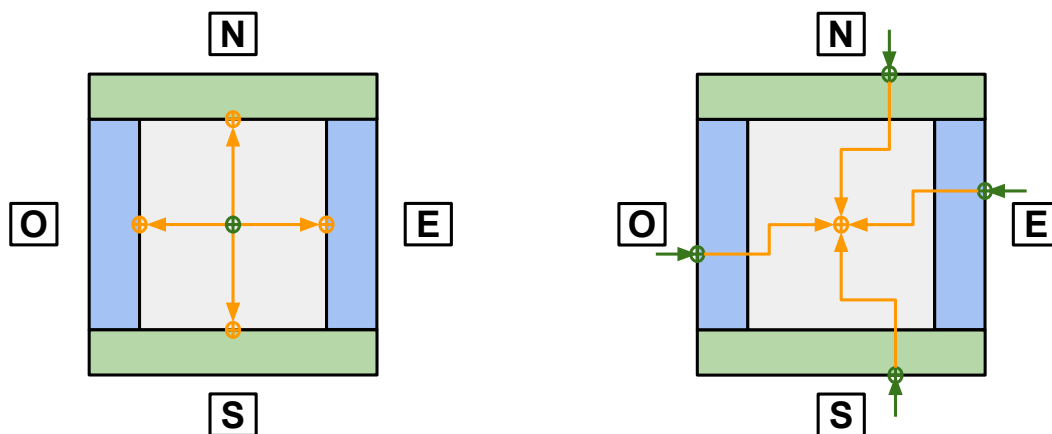


FIGURE 4.39 – Exemple de détection de présence de *VIAs*

Les explications d'estimation de longueur de fils et de coût de *VIA* ont été détaillées en prenant compte comme exemple un canal de routage horizontal ou un connecteur horizontal, la gestion et le raisonnement s'appliquent de façon similaire aux cas des canaux

de routage verticaux et connecteurs verticaux. Le traitement des espaces de routage de type *strut* est réalisé en fonction de leur forme. Lorsque leur largeur (resp. hauteur) est plus grande que leur hauteur (resp. largeur), le traitement est similaire à celui des canaux de routage horizontaux (resp. verticaux).

Dans le cas où le *Vertex* courant est numérique et le *Vertex* voisin est analogique, on emploie la méthode d'estimation de longueur de fils analogique. Pour cette utilisation, le point entrant et le point sortant d'un *Vertex* seront toujours le centre de la *GCell*. On estime que tout fil passant par des pavés numériques nécessite moins de précision en terme d'estimation de longueur de fils tandis que les modules de la partie analogique et les canaux de routage nécessitent davantage de précision. C'est pourquoi l'estimation de longueur avancée est limitée uniquement à la partie analogique. La figure 4.40 présente les situations rencontrées lorsqu'on se propage vers un *Vertex* numérique depuis un *Vertex* analogique et inversement.



(a) Propagation à partir d'un *Vertex* numérique vers un *Vertex* analogique

(b) Propagation vers un *Vertex* numérique à partir d'un *Vertex* analogique

FIGURE 4.40 – Gestion d'estimation de coût de longueur entre un *Vertex* numérique et un *Vertex* analogique

Implémentation de gestion de la mémorisation de la référence

Suite à l'estimation du coût vers le *Vertex* voisin, la mémorisation passe par un certain nombre d'étapes, en particulier si le *Vertex* courant est analogique. L'ensemble des étapes de la gestion de mémorisation de la référence est présenté par la figure 4.41 :

- **Comparaison avec le coût du *Vertex* voisin** : Le coût du *Vertex* voisin est valide si son attribut *_stamp* correspond à l'identificateur du *net* en cours *_netStamp*. Si son *_stamp* n'est pas à jour ou qu'il soit à jour et que le coût actuel est inférieur au coût du *Vertex* voisin, la condition "Ce coût est plus faible que celle du *Vertex* voisin ?" est remplie.
- **Le coût actuel est plus faible que le coût du *Vertex* voisin** : Cette étape implique une mémorisation du *Vertex* courant en tant que référence du *Vertex* voisin avec l'attribut *_from*. Si le *Vertex* voisin est analogique, il mémorise également l'occupation du fil passant par le *Vertex* courant en utilisant l'attribut *_intervfrom* de type *Interval*

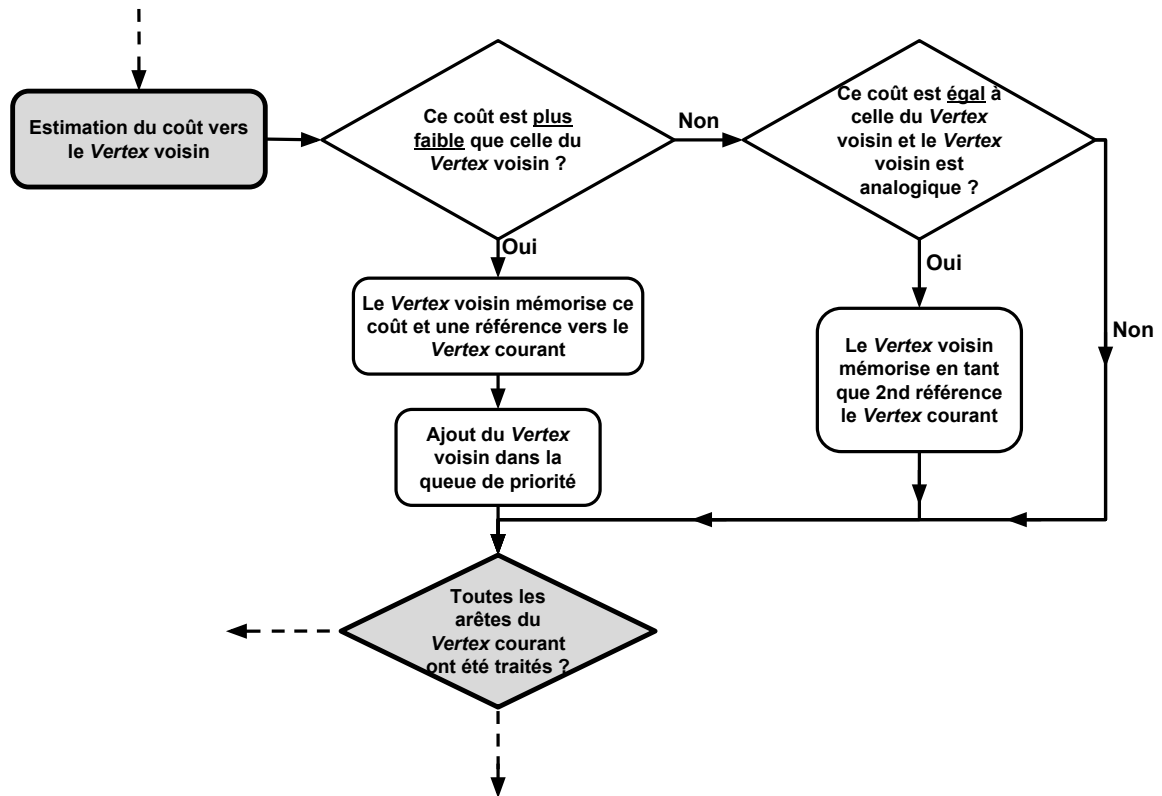


FIGURE 4.41 – Étapes de gestion de mémorisation de la référence

dans les informations complémentaires *_adata*. Le *Vertex* voisin est ensuite ajouté à la file de priorité ou bien remplacé, s'il était déjà présent, dans la file de priorité en fonction de son nouveau coût.

- **Le coût actuel est égal au coût du *Vertex* voisin** : Afin de gérer les erreurs d'estimation de coût de longueur mentionnées dans la partie 4.4.3, cette étape consiste à mémoriser une seconde référence. Le coût actuel doit être égal au coût du *Vertex* voisin, il est aussi nécessaire que le *Vertex* courant et la référence du *Vertex* voisin accèdent au *Vertex* voisin par le même côté. Si ces conditions sont remplies, le *Vertex* voisin mémorise l'occupation des fils passant par le *Vertex* courant en utilisant l'attribut *_intervfrom2* ainsi que la référence vers le *Vertex* courant avec l'attribut *_from2* dans les informations complémentaires *_adata*. Le *Vertex* voisin est déjà dans la file de priorité avec un coût identique et ne nécessite donc pas de remplacement dans la file de priorité.
- **Le coût actuel est supérieur au coût du *Vertex* voisin** : Cela implique que le chemin passant par le *Vertex* courant n'est pas optimal. L'algorithme passe au *Vertex* voisin suivant.

Dans la partie précédente, on a illustré les cas possibles d'occupation de fils avec l'exemple de la figure 4.37. Pour un canal de routage horizontal (resp. vertical), il est pertinent de connaître l'occupation du fil en abscisse (resp. ordonnée). Lorsque le routage global sera complété, l'information d'occupation de chacun des fils y passant sera étudiée afin de connaître la taille nécessaire au canal de routage pour que les fils puissent être effectivement routés.

La figure 4.42 présente l'utilisation de la classe *Interval* pour représenter l'occupation d'un fil. Les attributs *_min* et *_max* correspondent aux valeurs *xmin* et *xmax* (resp. *ymin* et *ymax*) d'occupation au sein du canal de routage horizontal (resp. vertical). L'attribut *_axis* permet de déterminer l'ordonnée (resp. l'abscisse) du fil dans le canal de routage horizontal (resp. vertical). Le traitement des espaces de routage de type *strut* est réalisé de façon similaire à la partie précédente, c'est-à-dire en fonction de leur facteur de forme.

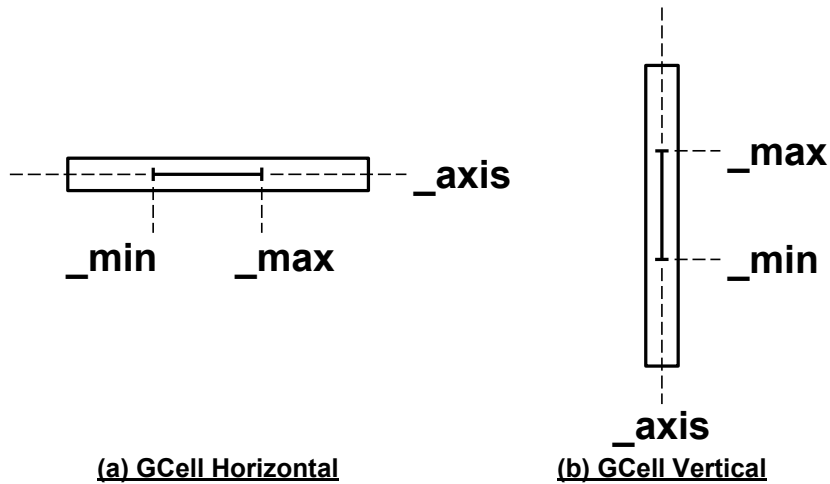


FIGURE 4.42 – Description de l'occupation d'un fil avec la classe *Interval*

La figure 4.43 suivante présente un exemple de chemin de *Vertex* ainsi que les intervalles d'occupation des fils. Les intervalles horizontaux sont représentés en vert foncé et les intervalles verticaux sont représentés en bleu foncé.

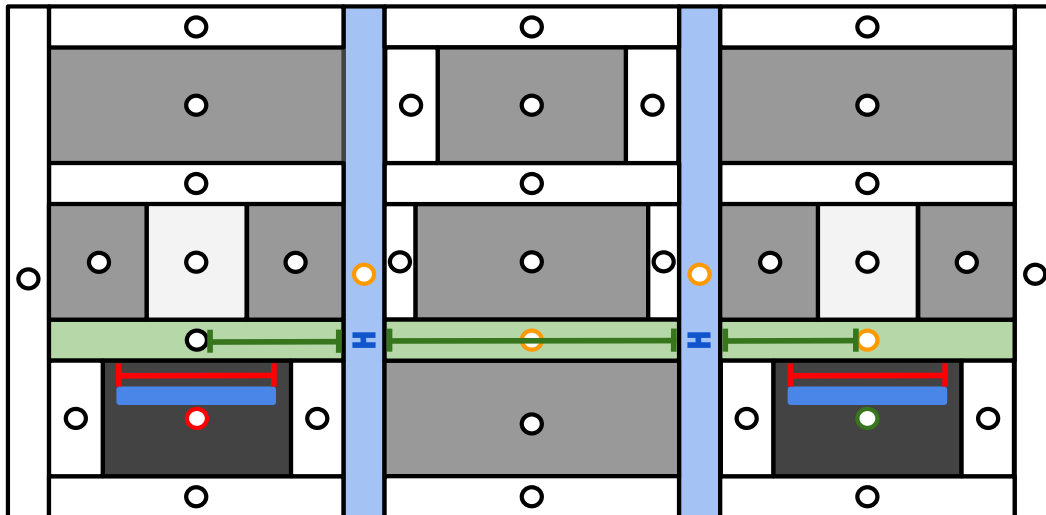


FIGURE 4.43 – Exemple de chemin de *Vertex* entre un *Vertex* source (sommet inférieur droit en vert) et un *Vertex* destination (sommet inférieur gauche en rouge)

Implémentation de la réinitialisation du graphe de routage

Une fois qu'un *Vertex* destination atteint "Le *Vertex* courant est un *Vertex* destination?" de la figure 4.35, le graphe de routage doit être réinitialisé si davantage de *Vertex*

destination doivent être atteints. Cette réinitialisation implique uniquement les *Vertex* composant le chemin de ce *Vertex* destination vers le *Vertex* source en suivant la gestion des composantes multiples de l'algorithme de la partie 4.4.2.

Pour l'ensemble de ces *Vertex*, qu'ils soient numériques ou analogiques, on réalise les trois actions suivantes :

- Remise à 0 de la valeur du coût *_distance*
- Ajout des *Vertex* à la liste des *Vertex* considérés comme *Vertex* source
- Insertion dans la file de priorité pour une nouvelle estimation des coûts vers les *Vertex* voisin

En plus de ces actions, les *Vertex* analogiques conservent les informations liées à l'occupation des fils. Chaque *Vertex* analogique contient les informations d'occupation des fils (*_intervfrom*) de sa référence (*_from*) et également les informations d'occupation des fils (*_intervfrom2*) de sa seconde référence (*_from2*) lorsqu'il en possède une. Chaque *Vertex* du chemin mémorise sa propre occupation des fils en mémorisant les informations avec l'attribut *_interv*.

Suite à la réinitialisation des *Vertex* du chemin, l'estimation analogique de coût d'arête utilise les informations d'occupation des fils précédemment mémorisées. Les points entrant du *Vertex* courant sont établis à partir de l'occupation des fils. À chaque nouvelle réinitialisation de chemin passant par des *Vertex* dont l'attribut *_interv* est déjà défini, l'intervalle *_interv* est étendu afin de considérer l'occupation des fils adéquate.

La figure 4.44 reprend l'exemple de la figure 4.43 et étendant l'arbre d'interconnexions à un nouveau sommet. Les intervalles en rouge représentent l'occupation de fils précédemment obtenue et le *Vertex* vertical central voit son intervalle (en bleu) d'occupation de fil être étendu suite à l'ajout du nouveau *Vertex* destination.

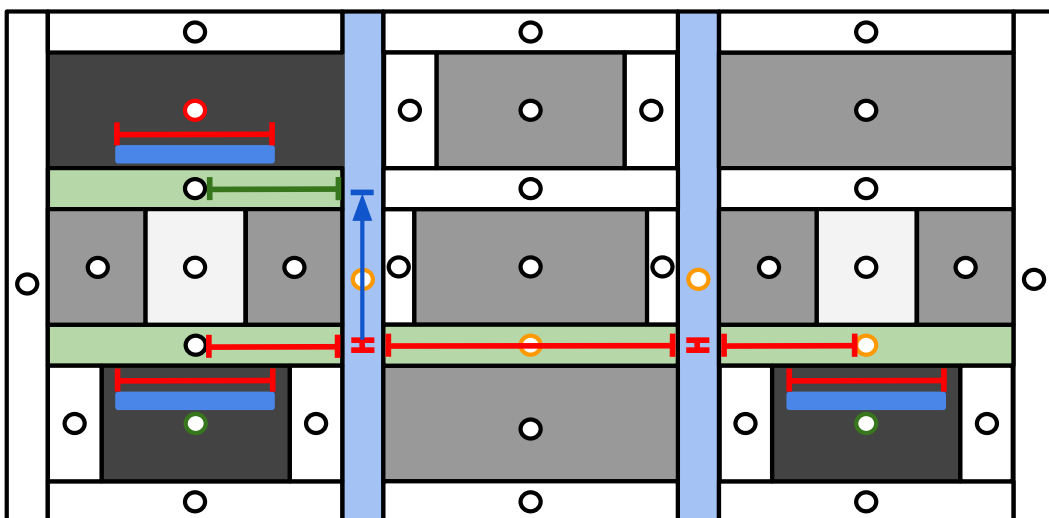


FIGURE 4.44 – Mise à jour des intervalles suite à l'atteinte d'un nouveau *Vertex* destination

4.5.5 Matérialisation des fils de routage global

Rappelons que l'algorithme 1 traite les signaux *net* par *net*. Une fois l'arbre d'interconnexions obtenu pour un *net* par l'algorithme de Dijkstra, le résultat est transformé en un ensemble de fils de routage. Cette étape, nécessaire à la mémorisation du résultat par le *net* donné, libère les ressources logicielles pour le traitement du prochain *net*. La matérialisation des fils de routage suit l'étape de la recherche d'arbre d'interconnexions pour un *net* donné. Ces fils de routage global constituent l'ensemble des informations qui seront transmises au routage détaillé afin de construire la topologie des fils de routage finaux. Cet ensemble d'informations est issu du résultat de l'algorithme de Dijkstra donnant l'arbre d'interconnexions pour le *net* qui vient d'être traité. Cette structure permet de mémoriser l'arbre d'interconnexions résultant et la libération des ressources utilisées par l'algorithme de Dijkstra pour une nouvelle itération.

Les fils représentant le routage global sont constitués de deux éléments provenant de la plateforme **Coriolis** :

- **Contact** : Un *Contact* est un objet représentant des points de contacts au sein d'un *net*. Dans ce contexte, un *Contact* représente un *VIA* permettant de joindre deux niveaux de métaux, en particulier les couches dédiées au routage global qu'on nomme *GMetalH* (Global Metal Horizontal) et *GMetalV* (Global Metal Vertical).
- **Segment** : Un *Segment* représente une couche de métal de forme rectangulaire et implicitement orientée. Un *Segment* de type horizontal (resp. vertical) a une longueur (resp. largeur) plus grande que sa largeur (resp. longueur) et utilise la couche de métal *GMetalH* (resp. *GMetalV*). Un *Segment* est associé à deux *Contacts* placés à ses extrémités.

Chaque *GCell* du pavage utilise un *Contact*, placé au centre, pour chaque utilisation du *Vertex* associé dans un arbre d'interconnexions d'un *net*. Une *GCell* aura donc n *Contacts* si elle fait partie de n *nets* différents. Pour représenter un arbre d'interconnexions, on utilise un *Segment* reliant deux *GCells* voisines. Le type du *Segment* dépend du type de l'arête joignant les deux *GCells* : on utilise un *Segment* horizontal (resp. vertical) lorsque l'*Edge* est de type horizontal (resp. vertical). Chaque création de *Segment* entraîne également une mise à jour de l'occupation des *Edges* concernés. Cela impacte en particulier le traitement des *nets* numériques tenant compte du coût de congestion.

La figure 4.45.(a) ci-dessus présente un exemple de matérialisation de fils de routage global pour un pavage numérique. On y observe les *Segments* horizontaux en vert, les *Segments* verticaux en violet et en jaune les *Contacts*.

Cette structure est employée de façon similaire pour les *nets* numériques et analogiques. Dans le cas des *nets* analogiques contenant une symétrie, la matérialisation des fils de routage est répliquée pour la partie symétrique. Que ce soit dans le cas d'une symétrie entre deux *nets* ou d'une symétrie où les sommets sont symétriques, l'arbre d'interconnexions obtenu suite à l'algorithme de Dijkstra représente la partie gauche (resp. basse) si l'axe de symétrie est vertical (resp. horizontal). Ainsi la matérialisation des *nets* symétriques implique la duplication des fils de routage pour l'autre partie de la symétrie.

La figure 4.45.(b) ci-dessus présente un exemple de matérialisation de fils de routage global pour un pavage analogique. On remarque que les *Contacts* ne sont pas toujours visuellement alignés. Chaque *Segment* connaît les deux *Contacts* qu'il relie d'un point

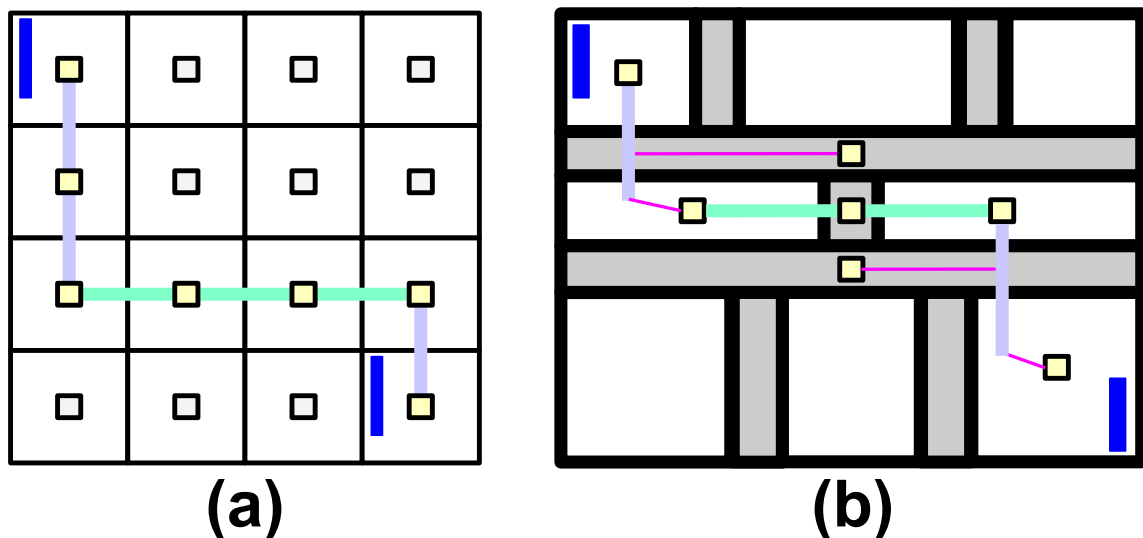


FIGURE 4.45 – Matérialisation des fils de routage global pour un exemple numérique (a) et un exemple analogique (b) pour un *net*. Les connecteurs à joindre sont représentés en bleu.

de vue logiciel, l'alignement n'est pas un problème et est pris en compte par le routeur détaillé. Au sein de la plateforme **Coriolis** des "lignes" roses qu'on nomme "*rubber*" permettent de montrer visuellement les relations de connexions des *Contacts* non alignés.

4.5.6 Mise à jour de l'occupation des fils de routage et redimensionnement des canaux de routage

Pour chaque *net*, la mise à jour de l'occupation des fils de routage et le redimensionnement des canaux de routage sont deux étapes destinées uniquement au circuit analogique. Après avoir obtenu l'arbre d'interconnexions d'un *net*, les informations d'occupation de fils sont récupérés au sein du *slicing tree*. L'occupation de fils de routage au sein d'un canal de routage est représentée sous la forme d'intervalles (voir figure 4.46). Les dimensions géométriques du canal définissent les extrémités que peuvent occuper les intervalles. Un intervalle représente l'occupation d'un fil de routage, cette occupation est déduite du chemin optimal obtenu lors de la construction de l'arbre d'interconnexions du *net*. Ces intervalles d'occupation sont mémorisés et le nombre maximum d'intervalles superposés déterminent la taille minimum nécessaire que le canal doit prendre. Pour un canal de routage horizontal (resp. vertical), on s'intéresse à l'occupation de fils en abscisse (resp. ordonnée). Avec le nombre m de pistes de routage utilisé, la hauteur (resp. largeur) d'un canal de routage horizontal (resp. vertical) sera de $m \times$ [taille d'une piste].

La figure 4.46 présente un exemple d'évolution d'occupation de fils au sein d'un canal de routage horizontal. Chaque sous-figure, de la figure (a) à la figure (d), correspond à un ajout d'une nouvelle occupation de fils d'un *net* dans le canal de routage. Le dernier intervalle ajouté est indiqué en vert, et le nombre d'intervalles occupant une plage de données est indiqué en bleu. Dans le cas de cet exemple, la plage de données ayant le plus d'occupation de fils est comprise dans la plage [3,4] comprenant trois fils.

Le redimensionnement des canaux de routage entraîne un renouvellement du place-

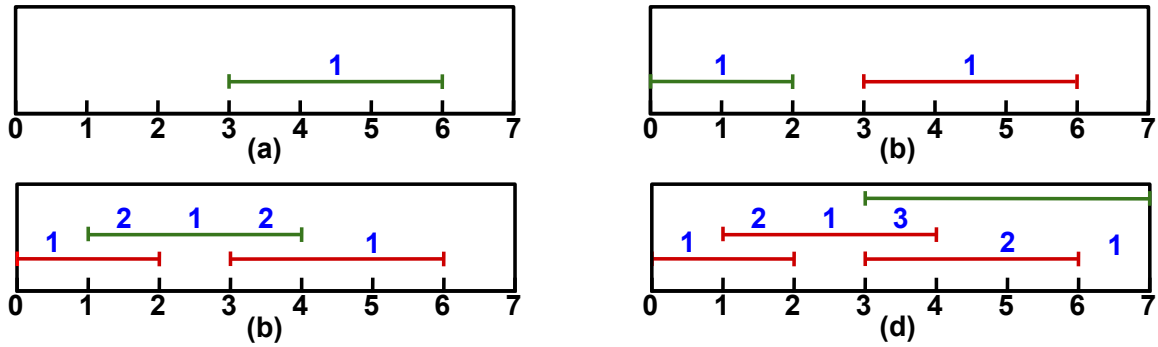


FIGURE 4.46 – Évolution de l’occupation de fils au sein d’un canal de routage horizontal après le traitement de quatre *nets* passant par cet canal de routage

ment des modules du circuit. Le placement modifié du circuit entraîne une mise à jour des informations des nœuds du *slicing tree* ainsi que du pavage avec la mise à jour des nouvelles positions et dimensions des *GCells*. Les *Contacts* associés aux *GCells* sont également replacés au centre de leur *GCell* correspondante. Ainsi, suite au remplacement des modules, l’ensemble des éléments *Contact* et *Segment* de type *GMetalH* et *GMetalV* est utilisé en entrée du routeur détaillé.

4.5.7 Le *slicing tree* durant la phase de routage global

Il existe un lien entre la structure donnée dédiée à la phase de routage et la phase de routage global. La création du pavage des espaces du circuit est réalisée en parcourant le *slicing tree* et dans lequel, on mémorise pour chaque nœud du *slicing tree* sa correspondance en tant que pavé. En particulier, cela permet au nœud du *slicing tree* d’accéder aux informations d’occupation de fils liées au routage d’un *net*. Ce lien permet également de mettre à jour les restrictions d’utilisation des *Vertex* (voir partie 4.5.3). La figure 4.48 présente l’implémentation des méthodes du *slicing tree* dédiées au routage global :

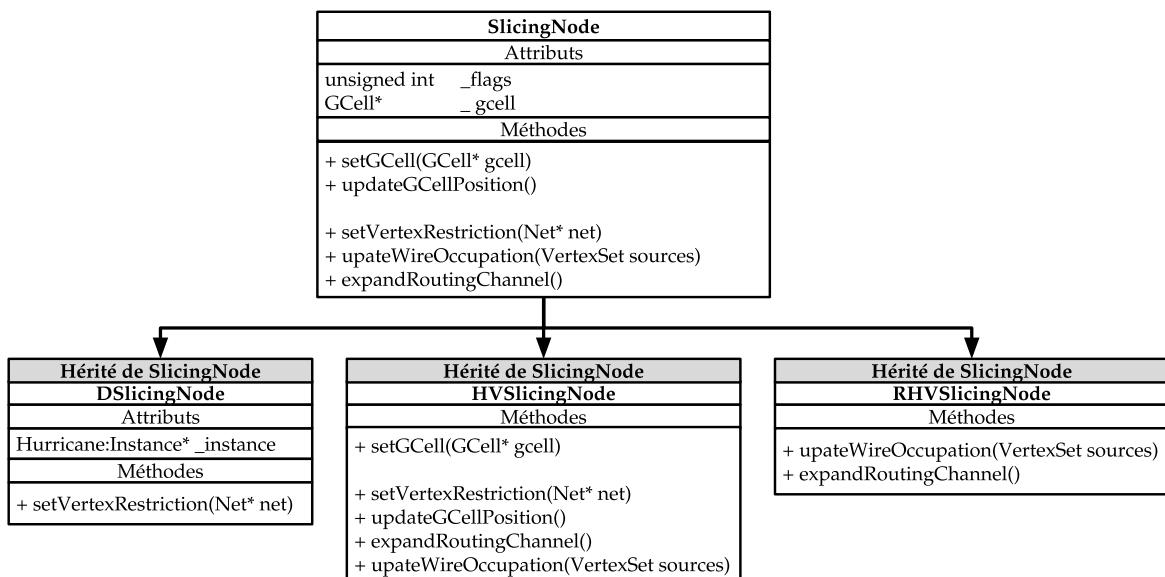


FIGURE 4.47 – Méthodes des classes du *slicing tree* dédiées à l’étape de routage global

- **Dans la classe *SlicingNode*** : En plus des attributs et des méthodes de la partie 3.5.1, cette classe possède un attribut *_gcell* stockant le lien vers le pavé *GCell* représentant le nœud. L'attribut *_flags* décrit précédemment contient également les restrictions d'utilisation du *Vertex* (plus de détails dans la classe *DSlicingNode*). La méthode *setVertexRestriction(Net* net)* permet d'imposer les contraintes de restrictions à l'attribut *_gcell*. La méthode *setGCell(GCell* gcell)* permet de transmettre les informations de position et de dimensions du nœud *SlicingNode* vers sa représentation en *GCell*. Lors d'un remplacement du circuit suite au dimensionnement des canaux de routage, les objets liées à la structure de données du routage (*GCell*, *Contact* et *Segment*) sont mis à jour par la méthode *updateGCellPosition()*. Les autres méthodes sont redéfinies dans les classes héritées.
- **Dans la classe *DSlicingNode*** : Pour restreindre un *Vertex* représentant un module, il est possible d'interdire le passage des fils pour chacun des quatre côtés d'un *Vertex* (est, ouest, nord et sud). La méthode *setVertexRestriction(Net* net)* consiste à restreindre l'attribut *_gcell*. Si le nœud possède un connecteur faisant partie du *net*, il est possible de restreindre des côtés du *Vertex* en fonction de la position du connecteur. Par exemple, si le connecteur est un segment horizontal se trouvant en haut du module, il est préférable de restreindre le côté sud afin de préférer un accès par les autres côtés. Les informations concernant les côtés à restreindre pour chaque connecteur sont contenues par l'attribut *_instance*.
- **Dans la classe *HVSlicingNode*** : La méthode *setGCell(GCell* gcell)* calcule les dimensions des canaux (modules et canaux de routage) du circuit dans le nœud hiérarchique courant et propage ces informations vers les nœuds fils. Les autres méthodes servent à parcourir le *slicing tree* pour l'application de ces méthodes au niveau des nœuds feuilles du *slicing tree*.
- **Dans la classe *RHVSlicingNode*** : La méthode *updateWireOccupation(VertexSet sources)* met à jour l'information d'occupation de fil du nœud *RHVSlicingNode* à partir du chemin de *Vertex* donné par l'argument *sources* après le traitement chaque *net* pendant le routage global. La méthode *expandRoutingChannel()* détermine le nombre maximum d'intervalles superposés d'occupation de fil des nœuds et met à jour la taille de le canal de routage.

Suite au redimensionnement des canaux de routage, le circuit est à nouveau placé avec les nouvelles dimensions des canaux de routage. Nous sommes conscients que le redimensionnement, ou le plus souvent l'agrandissement, des canaux de routage peut engendrer des différences d'arbres d'interconnexions de par le changement de position des modules et des canaux de routage du circuit. La recherche d'arbres d'interconnexions à partir de l'algorithme de Dijkstra reste une approximation des arbres d'interconnexions finaux constitués de vrais fils. Elle nous permet l'estimation d'une bonne solution et l'agrandissement des canaux de routage permet la faisabilité de cette solution ce qui est critère important dans l'utilisation d'une solution optimisée.

4.5.8 Description des contraintes de routage dans le script *Python*

Tout comme la description du placement à partir des scripts *Python* (voir partie 3.5.1), les contraintes de routage peuvent être décrites dans le même fichier script *Python*. Le concepteur a la charge de décrire la *netlist* en précisant les *nets* étant symétriques et le type de symétrie appliqué (symétrie entre deux *nets* ou symétrie entre les sommets d'un

unique *net*) ainsi que la largeur des fils de routage pour chaque *net*.

Dans la mesure où le circuit contient des symétries définies au sein du *slicing tree*, le redimensionnement se fait en respectant ces contraintes de symétries. Chaque canal de routage inclus dans une partie symétrique du circuit possède un canal de routage symétrique de l'autre côté de l'axe de symétrie. Le redimensionnement est réalisé en tenant compte de la taille minimum nécessaire pour l'occupation de fils requise entre les deux canaux de routage d'une paire symétrique.

La description des *nets* est présentée par le code 4.1 suivant :

Code 4.1 – Description des *nets* du circuit

```

1 self.netTypes = { 'net1': [ {W: value}
2                       , ('device1', 'PortA')
3                       , ('device2', 'PortB')
4                       , ...
5                       , ('deviceN', 'PortZ')
6                       ],
7                       .
8                       .
9                       .
10                      }
```

Un *net* est décrit à partir d'une paire clé-liste pour laquelle la clé représente le nom du *net* (dans le code 4.1, il s'agit de *net1*) et d'une liste contenant l'information des modules et de leurs connecteurs faisant partie du *net* (dans le code 4.1, (*device1*, *PortA*) correspond au connecteur *PortA* du module *device1*). Pour indiquer une taille de fil non minimale pour le *net*, le premier élément du dictionnaire doit être écrit avec "*W: value*" pour lequel *value* représente la largeur du fil de routage.

Les *nets* symétriques sont décrits au sein de la description d'un nœud hiérarchique (voir partie 3.5.1). La syntaxe de *nets* symétriques est présentée de la manière suivante avec le code 4.2

Code 4.2 – Description des *nets* du circuit

```

1 self.addSymmetryNet( NodeType, self.getNet('net1') )
2 self.addSymmetryNet( NodeType
3                       , self.getNet('net2'), self.getNet('net3')
4                       )
```

L'argument *NodeType* doit être remplacé par le type du nœud hiérarchique dans lequel est appliquée la symétrie, si le nœud hiérarchique est horizontal, il faut indiquer *HNode* et si le nœud hiérarchique est vertical, il faut indiquer *VNode*. Le type de symétrie considéré dépend du nombre de *nets* fournis en tant qu'argument. Dans cet exemple, le *net* "*net1*" est considéré comme étant un *net* dont les sommets sont symétriques et les *nets* "*net2*" et "*net3*" sont symétriques entre eux. La position de l'axe de symétrie est implicite et dépend du type du nœud hiérarchique. Si le nœud hiérarchique est horizontal (resp. vertical) alors l'axe de symétrie est horizontal (resp. vertical) et se trouve y (resp. x) = [position y (resp. x) du nœud hiérarchique + hauteur/2 (resp. largeur/2) du nœud hiérarchique].

4.6 Conclusion

Notre approche de routage des circuits analogiques et mixtes est composée de deux étapes : une étape de routage global et une étape de routage détaillé. L'étape de routage global a pour but de déterminer le parcours du chemin des fils pour tous les *nets*. Pour cela, le routage global débute en transformant le résultat du placement en un graphe de routage permettant de représenter les ressources et la topologie dédiée au routage. À partir de ce graphe de routage, un arbre d'interconnexions est recherché en utilisant l'algorithme de Dijkstra et suivant différentes contraintes telles que la minimisation de la longueur des fils ou le nombre de *VIA*s. De par leur topologie différente, les estimations de longueur sont adaptées en fonction du type de *net* traité (numérique ou analogique). Cette recherche d'arbres d'interconnexions peut être limitée à une zone afin de réduire le temps de calcul comme par exemple dans le cadre de traitement de *nets* numériques ou de *nets* analogiques symétriques.

On souligne que la structure de données a été pensée de telle sorte à ce que la phase de placement et la phase de routage communiquent des informations entre elles. La figure 4.48 présente les informations échangées entre les éléments du placement (*slicing tree*) et ceux du routage (pavage et graphe de routage).

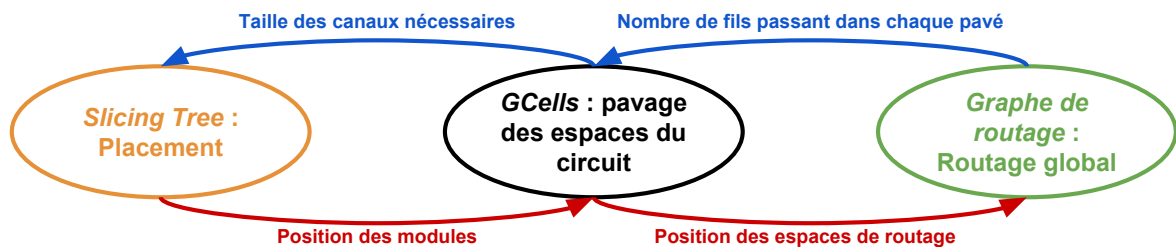


FIGURE 4.48 – Échanges d'informations entre la phase de placement et la phase de routage

Le *slicing tree* transmet la position des modules du circuit au routeur global. À partir de ces informations, le routeur global est capable de transformer le placement induit en un pavage composé de modules et de canaux de routage (*GCells*). Le pavage permet la création du graphe de routage permettant d'établir les arbres d'interconnexions pour chacun des *nets*. Ces informations d'interconnexions sont mémorisées au niveau des pavés faisant partie d'un arbre d'interconnexions, les pavés mémorisent le passage optimal des fils de routage après le traitement d'un *net*. Le *slicing tree* récupère les informations mémorisées au niveau des pavés pour mettre à jour les informations d'occupation de fils au sein du *slicing tree*, chaque canal de routage du circuit possède un nœud au sein du *slicing tree*.

Le résultat du routage global, pour chacun des *nets*, est représenté par une structure composée de contacts et de segments virtuels. Ces éléments virtuels permettent de déterminer le passage des fils d'un *net* à travers différentes régions du circuit pour l'étape de routage détaillé. La phase du routage détaillé a pour objectif de finir le routage en construisant les fils finaux, en les plaçant et en résolvant les situations de superpositions. Dans la continuité du routage global, le routage détaillé est mixte d'un point de vue traitement des fils de routage, c'est-à-dire que l'ensemble des fils de routage, qu'ils soient numériques ou analogiques, est traité selon un unique et même algorithme.

4.7 Références

- [1] Chris Chu. Flute : fast lookup table based wirelength estimation technique. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 696–701. IEEE Computer Society, 2004. [87](#)
- [2] Mohammad Torabi and Lihong Zhang. Efficient ilp-based variant-grid analog router. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, pages 1266–1269. IEEE, 2016. [87](#), [88](#)
- [3] Lihong Zhang Mohammad Torabi. A fast hierarchical adaptive analog routing algorithm based on integer linear programming. In *ACM Transactions on Design Automation of Electronic Systems*, 2017. [87](#), [88](#)
- [4] Jin-Tai Yan and Zhi-Wei Chen. Electromigration-aware rectilinear steiner tree construction for analog circuits. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pages 1692–1695. IEEE, 2008. [87](#)
- [5] Jin-Tai Yan and Zhi-Wei Chen. Obstacle-aware multiple-source rectilinear steiner tree with electromigration and ir-drop avoidance. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011. [87](#)
- [6] Changxu Du, Yici Cai, and Xianlong Hong. A performance driven probabilistic resource allocation algorithm for analog routers. In *Circuits and Systems, 2008. MWS-CAS 2008. 51st Midwest Symposium on*, pages 730–733. IEEE, 2008. [87](#)
- [7] Hung-Chih Ou, Hsing-Chih Chang Chien, and Yao-Wen Chang. Simultaneous analog placement and routing with current flow and current density considerations. In *Proceedings of the 50th Annual Design Automation Conference*, page 5, 2013. [87](#)
- [8] Hung-Chih Ou, Hsing-Chih Chang Chien, Yao-Wen Chang, 1Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, and Taiwan. Non-uniform multilevel analog routing with matching constraints. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, 2012. [87](#), [88](#)
- [9] IBM. Ilog cplex optimizer, 2012. [87](#)
- [10] Chia-Yu Wu, Helmut Graeb, Jiang Hu, Dept. of ECE, Inst. of EDA, and Dept. of ECE. A pre-search assisted ilp approach to analog integrated circuit routing. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*. IEEE, 2015. [88](#)
- [11] Chin Yang Lee. An algorithm for path connections and its applications. *IRE transactions on electronic computers*, (3) :346–365, 1961. [88](#)
- [12] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1) :269–271, 1959. [88](#)
- [13] Damien Dupuis. *Knik : Routeur global pour la plate-forme CORIOLIS*. PhD thesis, Paris 6, 2009. [90](#), [92](#), [94](#), [104](#), [111](#)

Chapitre 5

Routage détaillé

Sommaire

5.1 Introduction	130
5.1.1 Structure générale de l'algorithme	130
5.1.2 Structure générale de l'algorithme	130
5.2 Achèvement du routage global	131
5.2.1 Principe des fils articulés	131
5.2.2 Briques de construction	133
5.2.3 Calcul des contraintes des segments	134
5.2.4 Ensemble de segments alignés	135
5.2.5 Position optimale d'un méta-segment	136
5.2.6 Opérations d'assouplissement	137
5.2.7 Achèvement du routage d'une <i>GCell</i>	140
5.3 Résolution des superpositions	141
5.3.1 Mouvement atomique	142
5.3.2 Ordonnancement des segments	143
5.3.3 Traitement d'un évènement	143
5.3.4 Description simplifiée de la fonction de coût	144
5.3.5 Automate d'état d'assouplissement	145
5.4 Contraintes des signaux analogiques	148
5.4.1 Routage symétrique	148
5.4.2 Segments larges	150
5.5 Conclusion	152
5.6 Références	152

5.1 Introduction

Le résultat du routage global correspond, pour chaque *net*, à l'ensemble des *GCells* traversées par le *net*. Lorsqu'un *net* traverse plusieurs *GCells*, il est mémorisé comme étant un *net* global. Si un *net* global ne comporte pas de changement d'orientation (Horizontal/Vertical), le *net* est mémorisé par un segment global. Les *GCells* intermédiaires ne sont pas mémorisées pour le routage détaillé, mais lors du tracé effectif des fils par des segments, il faut vérifier que ceux-ci restent à l'intérieur des *GCells* issues du routage global. Ce cas se produit en particulier pour les circuits numériques.

Le routeur détaillé est le résultat d'années de travail dédiées dans un premier temps au routage détaillé de circuits numériques. Dans le cadre de cette thèse, la base de données du routeur détaillé a été enrichie afin de pouvoir incorporer le routage détaillé mixte et analogique. En particulier, les travaux de la thèse incluent la construction des topologies des fils de routage pour les circuits analogiques. L'ensemble de ce chapitre sur le fonctionnement du routeur détaillé a été co-rédigé avec Jean-Paul Chaput.

5.1.1 Structure générale de l'algorithme

La structure de données a principalement été construite en réponse aux problèmes suivants en séparant nettement les objectifs entre l'étape de routage global et celle du routage détaillé :

- **Le routage global** effectue une recherche de chemins (*maze routing*) jusqu'à un niveau de granularité (défini par la taille des *GCells*) très fin. Dans la zone numérique, la taille d'une *GCell* est approximativement carrée, la longueur du côté étant proche de la hauteur d'une *standard cell*. Une *GCell* contiendra donc entre une et trois *standard cell*. L'étape finale achevant le routage interne d'une *GCell* est donc très simple. Elle ne nécessite pas d'algorithme complexe.
- **Le routage détaillé** doit résoudre les superpositions entre segments en les déplaçant, sachant que le déplacement d'un segment conservera la connexité électrique du *net*. A cet égard, le routage détaillé peut être considéré comme un placeur. Une classe commune de routeur détaillé utilise une représentation matricielle de l'espace de routage. C'est-à-dire que chaque intersection entre une piste horizontale et une piste verticale est associée à un point de la matrice. Cette représentation occupe en mémoire une taille proportionnelle à la surface du circuit divisé par le pas de routage. Pour des circuits de taille importante, l'empreinte mémoire devient problématique. Une alternative est d'utiliser le *line probing* [1] [2].

5.1.2 Structure générale de l'algorithme

À l'issue de l'étape de routage global, nous disposons, pour chaque signal (*net*), d'un ensemble de segments connectant les *GCells* contenant ses terminaux. Le routage détaillé consiste à créer les connexions jusqu'aux terminaux, sans court-circuit, entre les différents signaux passant dans une *GCell*.

L'étape de routage détaillée peut grossièrement être décomposée en deux phases :

1. **L'achèvement du routage d'un signal** : On crée une solution optimale du signal, sans considération des autres *nets*, dans chaque *GCell*.

2. **La résolution des superpositions entre les différents segments des différents signaux** : On utilise pour cela un algorithme de type *rip-up and reroute*.

5.2 Achèvement du routage global

5.2.1 Principe des fils articulés

Dans un circuit intégré, un fil électrique, réalisation physique d'un signal, est construit à partir de segments horizontaux et verticaux et de *VIA*s assurant une connexion entre deux couches métalliques adjacentes. Dans cette section, nous introduisons la structure de données nous permettant de modéliser ces objets, les règles que nous leur appliquons et les propriétés qui en découlent. Les figures 5.1 et 5.2 présentent la transformation d'un routage global en un routage détaillé pour un signal donné.

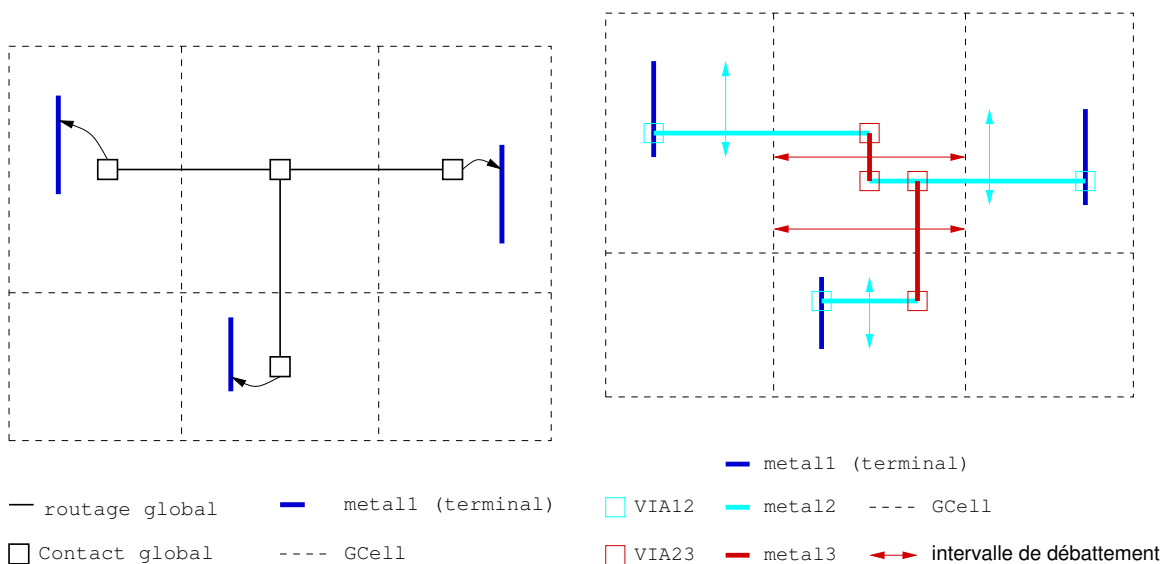


FIGURE 5.1 – Exemple de fil électrique, routage global

FIGURE 5.2 – Exemple de fil électrique, routage détaillé

Définition : Un assemblage déterminé de segments et de contacts implémentant un fil électrique sera appelée une **topologie**.

Pour une topologie donnée, l'ensemble des positions des axes des segments suffit pour définir son placement complet. Le déplacement des axes est limité à un intervalle de débattement ou contrainte garantissant la connexité électrique (pas de coupure dans les fils). L'extension des segments est entièrement déterminée par la position des perpendiculaires à ses extrémités. Dans la suite, lorsqu'on parlera de la position d'un segment, **on se référera toujours à celle de son axe**.

Une topologie peut ne pas présenter assez de degrés de liberté pour pouvoir faire un placement sans recouvrement d'autres fils. Dans ce cas, on procède de façon contrôlée à un assouplissement, cf. 5.2.6.

Définition : Un contact est dit ponctuel si, la surface définie par l'intersection des axes des segments incidents se réduit à un point. Exemple : Les contacts 5.3.a et 5.3.b ne sont

pas ponctuels, 5.3.c l'est.

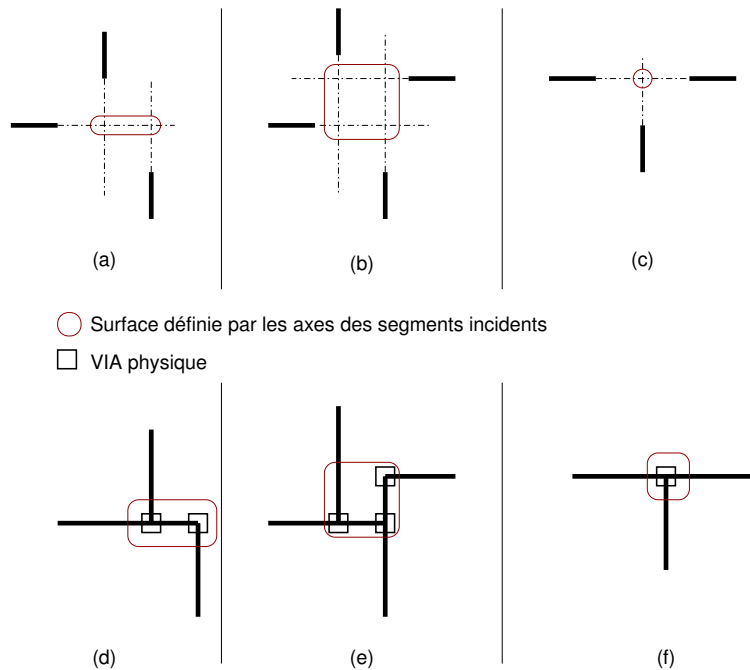


FIGURE 5.3 – Exemple de contacts non-ponctuels et ponctuels

Dans le cas de contacts non-ponctuels il est possible de rétablir la connexion en allongeant les extrémités des segments et en mettant plusieurs *VIA*s physiques comme on peut le voir en 5.3.d et 5.3.f. Les problèmes liés à la gestion de ces types de contacts apparaissent ici :

1. À un contact va correspondre entre un (cas 5.3.c) et trois (cas 5.3.e) *VIA*s physiques.
2. Pour assurer la connexion électrique, les extrémités des segments incidents doivent être allongées ou raccourcies, et dans le cas 5.3.e plusieurs solutions équivalentes sont même possibles.
3. Enfin, à chaque fois que l'axe d'un segment est déplacé les extensions des segments peuvent bouger et le nombre de *VIA*s physique varier.

Pour rendre l'algorithme de routage plus efficace, on va simplifier la gestion de la structure de données en se restreignant aux contacts ponctuels uniquement. Avec pour conséquences :

- Pour chaque contact, on aura un unique *VIA* physique associé.
- Le déplacement de l'axe d'un segment n'entraîne jamais de modification de la longueur de ses extrémités. Seules les extrémités des perpendiculaires sont allongées/-rétrécies. On dit que le segment coulisse.

Les contacts non-ponctuels peuvent être décomposés en contacts ponctuels comme illustré en figure 5.4.

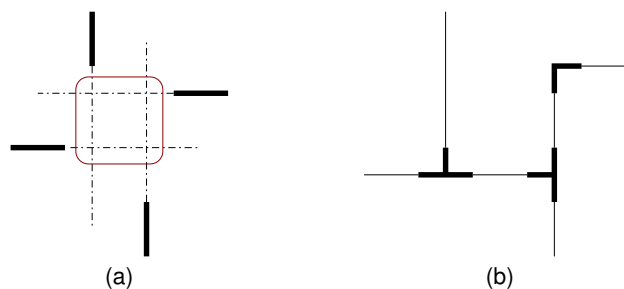


FIGURE 5.4 – Exemple de décomposition en contacts ponctuels

5.2.2 Briques de construction

Pour construire les fils d'un signal, on utilise deux types d'élément :

- **Les segments** : Horizontaux ou verticaux, ils connectent deux contacts *source* et *target* entre eux. Les coordonnées (x,y) de la *source* sont toujours inférieures ou égales à celle de la *target*. Les segments peuvent être globaux (issus du routage global) si la *source* et la *target* n'appartiennent pas à la même *GCell* ou peuvent être locaux.
- **Les contacts** : Ils servent à articuler les segments entre eux. Quatre types de contacts sont disponibles : le contact *terminal* (figure 5.5), le coude (*Turn*, figure 5.6), la branche horizontale (*HTee*, figure 5.7) et la branche verticale (*VTee*, figure 5.8). Les segments doivent toujours être placés de façon à ce que le contact soit ponctuel, voir en particulier le cas des *HTee* et *VTee*.

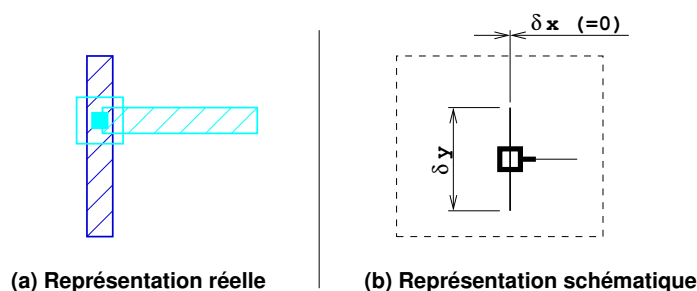
Une zone de contrainte est associée à chaque contact. Elle indique le rectangle dans lequel le centre du contact doit se trouver.

Un contact est aussi associé à une *GCell* qui définit la zone maximale de contrainte. C'est à dire qu'un contact ne peut être placé que dans sa *GCell*.

Un contact est, au plus bi-métallique, c'est à dire qu'il connecte deux niveaux de métaux adjacents. Par exemple, un contact en *layer VIA23* ne pourra être associé qu'à des segments en *METAL2* ou *METAL3*.

Le contact *terminal*

Il assure la connexion entre **un** segment et le *terminal* d'un signal. La zone de contrainte est la surface du *terminal* (réduite de la demi-taille du contact).

FIGURE 5.5 – Schéma du contact *terminal*

Le coude (*Turn*)

Il assure la connexion entre **deux** segments perpendiculaires. Les coordonnées x et y du contact sont déterminées par, respectivement, l'axe du segment vertical et l'axe du segment horizontal. La zone de contrainte est la surface de la *GCell* auquel appartient le contact.

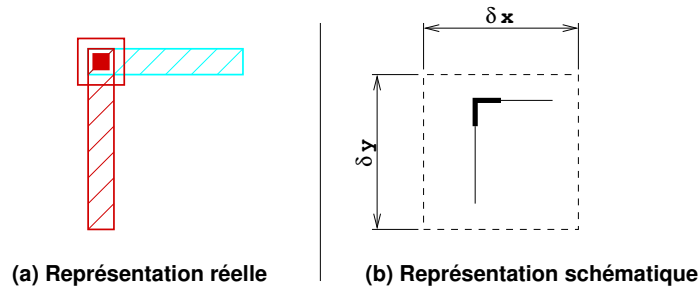


FIGURE 5.6 – Schéma du coude (*Turn*)

La branche horizontale (*HTee*)

Il assure la connexion entre **deux** segments horizontaux et **un** segment vertical. Pour que le contact soit ponctuel, cela impose que les deux segments horizontaux soient maintenus alignés. La zone de contrainte est celle de la *GCell*.

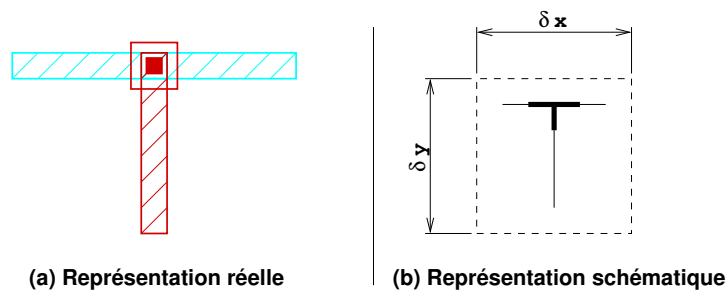


FIGURE 5.7 – Schéma de la branche horizontale (*HTee*)

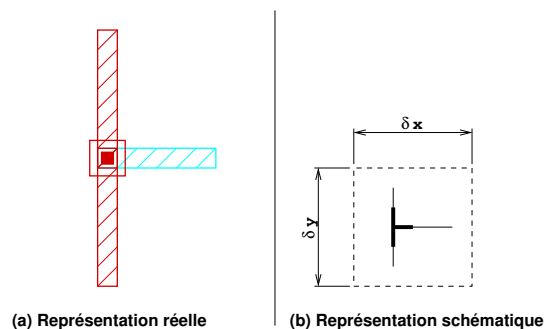
La branche verticale (*VTee*)

Pendant vertical du *HTee*, il assure la connexion entre **deux** segments verticaux et **un** segment horizontal. Pour que le contact soit ponctuel, cela impose que les deux segments verticaux soient maintenus alignés. La zone de contrainte est celle de la *GCell*.

5.2.3 Calcul des contraintes des segments

La contrainte d'un segment est l'intervalle dans lequel peut se déplacer son axe.

Les contraintes des segments sont déduites des contacts auxquels ils sont connectés et, dans le cas des segments globaux, des *GCells* qu'ils traversent. Sur la figure 5.9, le déplacement vertical de l'axe du segment "id :50" n'est pas seulement limité par ses contacts

FIGURE 5.8 – Schéma de la branche verticale (*VTee*)

source (dans la *GCell a*) et *target* (dans la *GCell e*), mais aussi par la *GCell c*.

Pour limiter la taille mémoire de la structure de données, les *GCells* gèrent une liste des segments les traversant mais les segments ne connaissent que leur intervalle de contrainte. Celui-ci est calculé une fois pour toute à leur création.

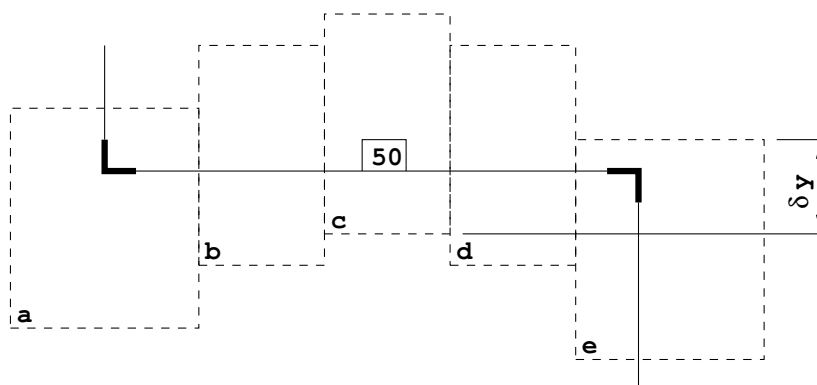


FIGURE 5.9 – Contrainte des segments globaux

5.2.4 Ensemble de segments alignés

Nous avons spécifié en 5.2.2 pour les contacts de branche horizontale (*HTee*) que les deux segments horizontaux attachés à la branche devaient obligatoirement rester alignés (positionnés sur le même axe). La conséquence de cette contrainte est que les segments horizontaux liés entre eux par des branches horizontales forment un ensemble aligné.

Pour simplifier la gestion de la structure de données, plutôt que d'autoriser le déplacement de n'importe lequel des segments de l'ensemble aligné, on en distingue un, le **représentant canonique** qui sera toujours utilisé pour déplacer l'ensemble. Le représentant canonique porte en outre des informations résumant l'ensemble, comme par exemple les positions *x* des *sources* et *targets*. Pour choisir le représentant canonique de façon déterministe, on prend le segment ayant l'identificateur (*id*) le plus petit.

Cette mécanique est transposée aux segments verticaux reliés par des *VTee*.

Définition : Un ensemble de segments alignés par des *VTees* ou des *HTees* forment un **méta-segment**. Les méta-segments sont manipulés au travers de leur représentant canonique.

L'exemple en figure 5.10 montre l'ensemble de segments alignés "id :51", "id :54" et "id :57". Le segment "id :51" en est le représentant canonique (trait doublé).

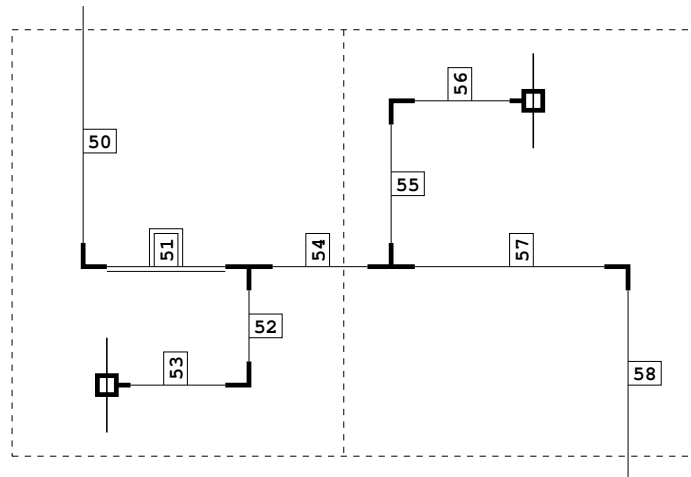


FIGURE 5.10 – Exemple d'ensemble de segments alignés

5.2.5 Position optimale d'un méta-segment

On définit la position optimale d'un méta-segment comme celle qui minimise la longueur des segments qui lui sont directement perpendiculaires.

Définition : Un **attracteur** est une coordonnée vers laquelle est attirée un segment. Dans le cas d'un méta-segment horizontal, les attracteurs seront déduits des segments perpendiculaires (verticaux) de la façon suivante :

- Si le perpendiculaire est *global*, l'attracteur sera la coordonnée y de son point d'intersection avec le bord de la *GCell*.
- Si le perpendiculaire est *local* (entièrement contenu dans une *GCell*) alors il est purement et simplement ignoré.
- Si le perpendiculaire est relié à un *terminal*, les deux coordonnées y des extrémités du *terminal* sont ajoutées à la liste des attracteurs (voir la figure 5.12).

Pour les segments verticaux il suffit de permuter coordonnées et directions.

À partir de la liste des attracteurs, la fonction d'attraction se calcule avec la formule suivante :

$$attraction(axis) = \sum_i |axis - attractor_i| \quad (5.1)$$

La fonction d'attraction est une composante de la fonction de coût utilisée pour le placement des méta-segments. **Remarque :** Dans les figures 5.11 et 5.12, l'échelle des pentes n'a pas été respectée.

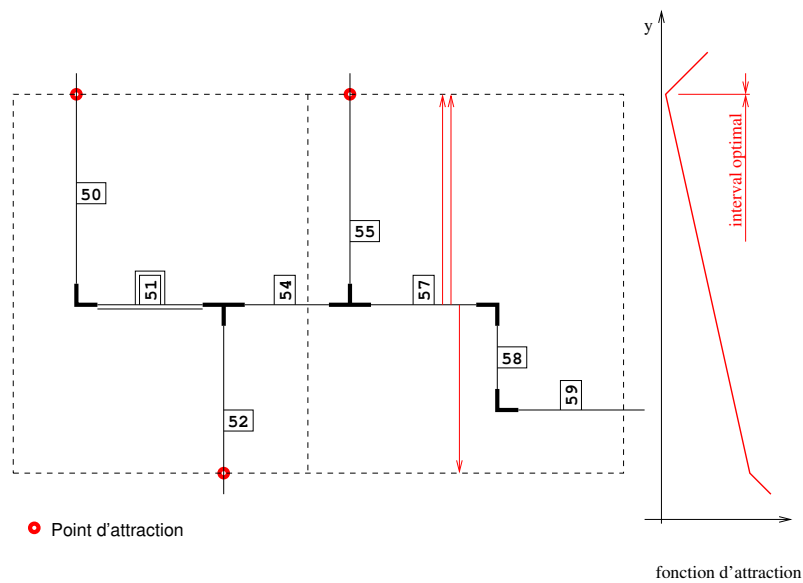


FIGURE 5.11 – Intervalle optimal, exemple avec perpendiculaires globaux seulement

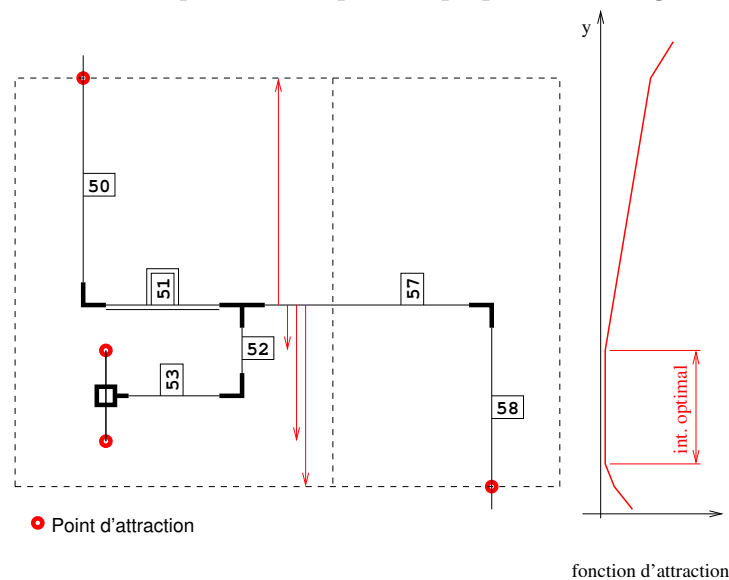


FIGURE 5.12 – Intervalle optimal, exemple avec perpendiculaires globaux et terminal

5.2.6 Opérations d'assouplissement

Les topologies initialement retenues pour les signaux utilisent le plus petit nombre de contacts possible, compte tenu des topologies admissibles. Lors de la phase de résolution des superpositions, la rigidité des topologies initiales peut s'avérer trop grande. Il est alors nécessaire de procéder à un assouplissement de la topologie. Pour cela nous disposons de deux transformations :

- La création de doubles coudes ou *doglegs*.
- Le changement de *layer*, si le nombre de métaux disponibles dans la technologie le permet (plus de deux niveaux de métallisation dédiés au routage).

Création de *dogleg*

La figure 5.13 explicite le processus de création d'un *dogleg* sur le segment "id :54". Il est strictement ordonné comme suit afin de garantir le déterminisme de la structure de données :

1. Le contact *target* est détaché du segment "id :54".
2. Deux nouveaux contacts ("d11" et "d12") de type *Turn* sont créés.
3. Le segment "id :54" voit sa *target* accrochée à "d11".
4. Le segment "id :70" perpendiculaire est créé entre "d11" et "d12".
5. Le nouveau segment parallèle "id :71" est créé entre "d12" et l'ancienne *target* du segment "id :54".

Le *layer* du segment perpendiculaire "id :70" sera celui immédiatement connexe à celui du segment "id :54" par en dessus ou par en dessous suivant le contexte.

En outre, on constate que le méta-segment composé de {"id :51", "id :54", "id :57"} se trouve coupé en deux. Après la création du *dogleg*, nous obtenons le méta-segment {"id :51", "id :54"} (canonique : "id :51") et {"id :71", "id :57"} (canonique : "id :57"). Dans le cas du second méta-segment, on notera que le segment canonique n'est pas forcément le premier dans l'alignement (c'est celui qui a l'identificateur le plus faible).

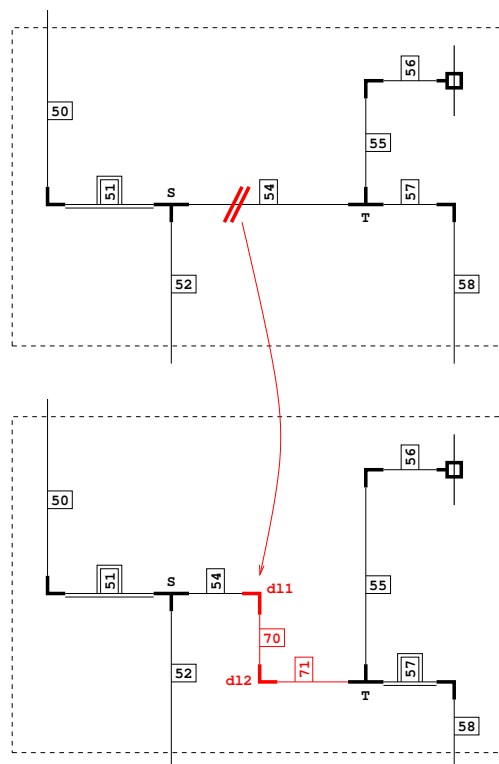
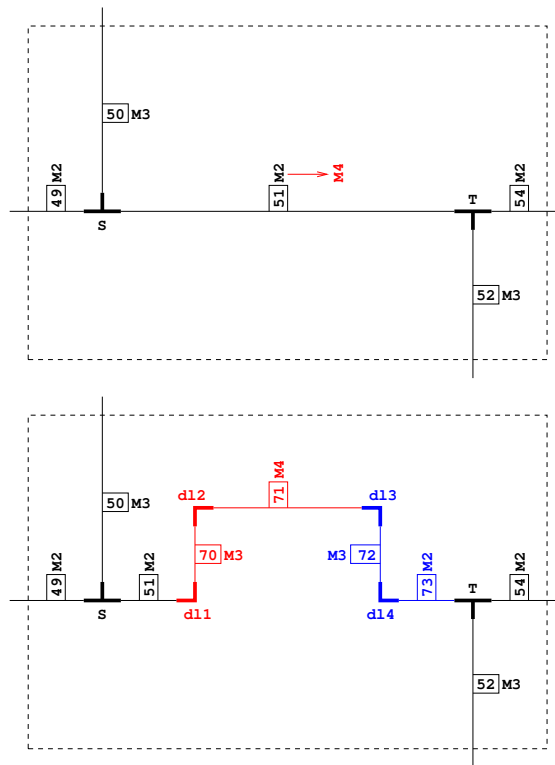


FIGURE 5.13 – Exemple de création de *dogleg*

Changement de *layer*

Sur la figure 5.14, le segment "id :51" voit son *layer* modifié de *METAL2* à *METAL4*, c'est à dire qu'on le remplace par un *layer* de niveau supérieur. Le processus détaillé est le suivant :

FIGURE 5.14 – Exemple de changement de *layer*

1. Le *layer* du segment "id :51" est changé.
2. Suite au changement de *layer*, il n'est plus certain que les contacts *source* et *target* du segment soient toujours électriquement connexes. Ils sont donc tous les deux invalidés topologiquement.
3. On revalide topologiquement les contacts *source* et *target* (l'ordre de revalidation est garanti déterministe).

Le mécanisme de revalidation topologique d'un contact est le suivant :

- Pour les contacts ne pouvant, au plus, que connecter deux *layers* adjacents, on vérifie que c'est toujours le cas après changement de *layer*. Par exemple, le contact *source* comporte maintenant les *layers* {METAL2, METAL3, METAL4}, dont METAL2 et METAL4 qui ne sont pas adjacents.
- Sur le segment "id :51", nous allons créer un *dogleg* qui va permettre de restaurer la connectivité. Les nouveaux contacts "dl1" et "dl2" permettront d'atteindre le METAL4 et dans la bonne direction (segments rouges).
- L'opération devant être répétée sur le contact *target*, il donnera naissance à un second *dogleg* et aux contacts "dl3" et "dl4" (segments bleus).

La revalidation topologique d'un contact ne nécessite que le mécanisme déjà vu de création des *doglegs*, ce qui simplifie la gestion de la structure de données.

Remarque : En raison de la méthode de création des *doglegs*, le segment qui va réellement être migré en METAL4 ne sera pas celui sur lequel la requête a été faite "id :51" mais "id :71", résultat des *doglegs*.

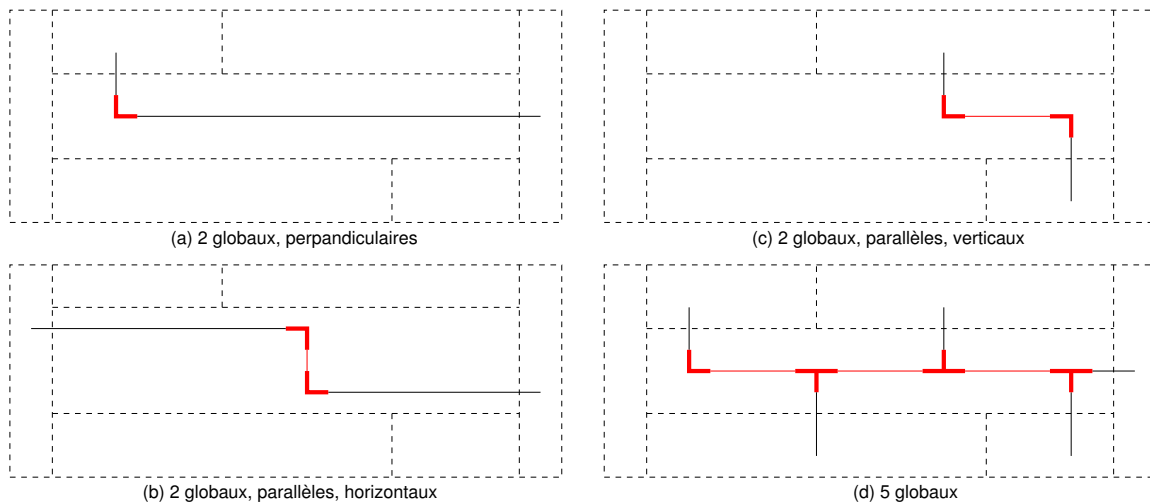


FIGURE 5.15 – Achèvement du routage d'un canal horizontal

5.2.7 Achèvement du routage d'une *GCell*

Le routage global d'un *net* est un arbre. On choisit une racine pour parcourir une par une chaque *GCell* de l'arbre avec un algorithme classique en profondeur d'abord en complétant le routage. Pour garantir le déterminisme, entre toutes les racines possibles, on choisira celle se trouvant le plus en bas à gauche.

Pour achever le routage d'un *net*, on distingue les 3 cas de *GCell* :

- *GCell* canal (Horizontal ou Vertical)
- *GCell* strut
- *GCell* device

Le cas du *strut* étant traité comme celui du canal, on illustre dans la suite le cas d'un canal horizontal et le cas du *device*.

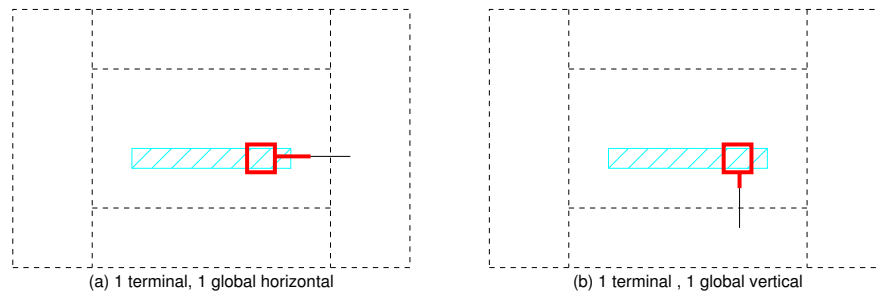
Cas du canal horizontal

Une *GCell* est reconnue comme un canal horizontal lorsqu'elle n'a qu'une voisine sur les faces Est/Ouest et plus d'une sur les faces Nord/Sud.

Dans les différentes configurations présentées figure 5.15, les segments noirs sont issus du routage global (ils sont à cheval sur au moins deux *GCells*) et les segments rouges sont ceux créés pour achever le routage de la *GCell*. Les configurations sont ordonnées par nombre de segments globaux croissants et complexité.

Cas d'un *device*

Une *GCell* de type *device*, par construction, est entourée d'une seule voisine sur chacune de ses faces. Pour un signal donné, une *GCell* *device* ne comportera qu'un seul connecteur. L'achèvement du routage dans ces conditions est extrêmement simple comme le montre la figure 5.16.

FIGURE 5.16 – Achèvement d'un *device*

5.3 Résolution des superpositions

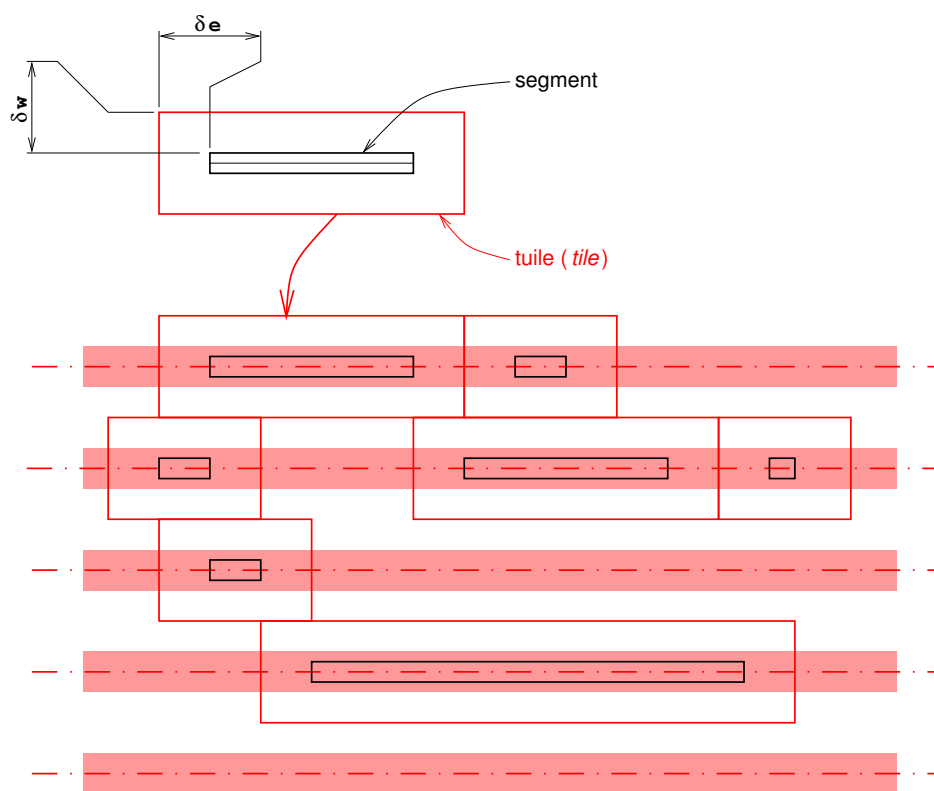


FIGURE 5.17 – Structure des pistes de routages

Important : Dans cette phase, nous ne considérons plus les segments individuels, mais les ensembles de segments alignés ou méta-segments. Même les segments uniques (non-alignés avec d'autres) seront représentés par des méta-segments. Dans la suite de cette partie, on utilisera de façon interchangeable les termes segment et méta-segment.

Jusqu'à présent, l'algorithme de routage détaillé, reposant sur la structure de données des segments articulés, ne requiert aucune information sur les pistes de routage. Seulement les directions préférentielles des métaux de routage.

Pour la résolution des superpositions, les pistes de routage sont créées et l'objectif de cette phase est d'assigner chaque segment à une piste. Dans une piste aucun recouvrement entre segments appartenant à des signaux différents n'est toléré. Un segment peut

donc se trouver dans deux états :

- **Placé**, c'est à dire assigné à une piste.
- **Non placé**, indépendant de toute piste.

L'algorithme de résolution des superpositions est du type *rip-up and re-route*, et les éléments manipulés sont les segments. Il est important de bien réaliser que la structure de données des pistes n'est pas prévue pour faire des recherches de chemins (*maze routing*). Elle vise à pouvoir faire des recherches rapides, dans le plan, pour savoir si une surface rectangulaire est libre ou non. C'est ce choix d'organisation du plan qui fait que le routeur est sur grille. Il serait parfaitement envisageable d'utiliser d'autres découpages du plan autorisant le routeur à devenir *gridless*.

Définition : L'algorithme de **rip-up en reroute** consiste à router les segments des *nets* du circuit selon un premier ordonnancement. Dans la mesure où des segments des *nets* ne peuvent être routés suite à cet ordre de traitement, certains segments routés sont invalidés selon des critères définis. Les segments restant non-routés sont alors routés à nouveau selon un nouvel ordonnancement. Ce processus est répété jusqu'à que tous les *nets* soient routés ou que l'algorithme estime que le circuit n'est pas routable.

La figure 5.17 montre comment un segment est transformé en une tuile (*tile*). Les tuiles sont construites de façon à ce que si deux tuiles sont adjacentes, les segments qu'elles représentent ne provoqueront aucune violation des règles de *DRC*. Sur cette même figure :

- La longueur δw est fixe, elle est déduite de la distance minimale de piste à piste.
- La longueur δe varie suivant le type de *VIA* situé à chaque extrémité du segment.

5.3.1 Mouvement atomique

L'opération de base de placement consiste à choisir la position de l'axe d'un segment pour l'aligner sur une piste dans laquelle il y a un espace suffisant pour insérer le segment / méta-segment. Les types de contacts que nous avons utilisés pour construire les segments articulés garantissent que quelque soit la position de son axe, les extensions du segments ne changent pas, voir la figure 5.18.

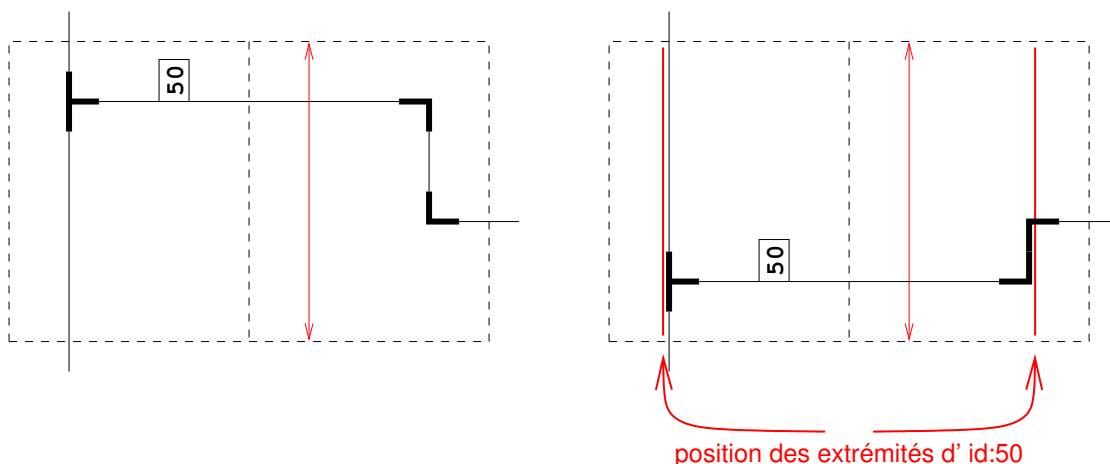


FIGURE 5.18 – Coulissage d'un segment

5.3.2 Ordonnancement des segments

Pour router les segments on utilise des évènements. Un évènement se compose d'une priorité, d'un segment et d'un état (cf. 5.3.4).

Les évènements sont placés dans une queue de priorité pour le placement. La priorité se compose de deux termes triés dans l'ordre suivant :

1. **Critère n°1 : Un niveau d'évènement**, un entier de 0 à 5 qui permet de forcer un ré-ordonnancement temporaire. Par exemple, lorsque l'on souhaite impérativement placer les perpendiculaires d'un segment avant celui-ci.
2. **Critère n°2 : Le degré de liberté d'un segment**, c'est le produit de la longueur du segment par son intervalle de contrainte (cf. 5.2.3).

$$priority = (longueur + 1) \times (longueur(contrainte) + 1) \quad (5.2)$$

On ajoute 1 à chaque terme afin qu'aucun des deux ne soit jamais nul. Le tri se fait par ordre croissant, 3 est plus prioritaire que 14.

3. **Critère n°3 : En cas d'égalité de tous les autres critères de tri**, on utilise l'identifiant du segment en ordre croissant, 37 est plus prioritaire que 38.

Dans la suite, nous écrirons un évènement en utilisant la convention suivante :

Event queue :

```

[0000] event( 5, 64, seg_id:25 )
[0001] event( 0,  3, seg_id:40 )
[0002] event( 0, 14, seg_id:33 )
[0003] event( 0, 65, seg_id:37 )
[0004] event( 0, 65, seg_id:38 )
|           | |           |
|           | |           +---- Identifiant du segment
|           | +----- Degré de liberté (priority)
|           +----- Niveau d'évènement
|
+----- Position de l'évènement dans la queue

```

Le prochain évènement à être traité est celui en position 0000.

Terminaison de l'algorithme : Il s'arrête lorsque la queue de priorité est vide. Sachant que le nombre de *doglegs* que l'on peut créer est limité et que le nombre de fois qu'un évènement peut être remis dans la pile est aussi borné, il est trivial de montrer que le programme s'arrêtera toujours.

Ordonnancement quasi-statique : Si l'on excepte le niveau d'évènement et que l'on considère que la longueur des segments varie peu, la priorité d'un segment va donc pas ou peu varier. Cela signifie que l'ensemble des segments sera routé dans un ordre déterminé dès le départ. La variation de longueur et le niveau d'évènement n'introduiront que des variations locales dans cet ordonnancement.

5.3.3 Traitement d'un évènement

Rip-up count : À chaque fois qu'un évènement est traité, son *rip-up* count est incrémenté, et ce, quelque soit l'issue du traitement. Le traitement d'un évènement consiste

à essayer de placer le segment auquel il est associé. Nous déterminons l'ensemble des pistes candidates situées dans l'intervalle de contrainte de l'axe du segment. Pour chacune de ces pistes nous calculons le coût d'insertion (la fonction de coût est détaillée dans la section 5.3.4). L'algorithme va alors essayer d'insérer le segment dans la candidate de plus faible coût. Quatre cas se présentent :

1. La piste est libre, on y insère le segment. L'évènement n'est pas remis dans la queue.
2. La piste est complètement ou partiellement obstruée, des actions d'insertion sont émises (i.e. d'autres évènements visant à faire bouger les segments obstrués). L'évènement courant est lui aussi remis dans la queue.
3. Le segment a atteint le nombre maximal de *rip-up* autorisés, auquel cas il est assoupli (voir section 5.3.5).
4. Le segment a atteint le nombre maximal de *rip-up* et il se trouve au dernier état d'assouplissement. Le routage de ce segment a échoué, il est définitivement sorti de la queue. Et le routage du circuit est en échec.

Considération sur l'ordonnement : Si le traitement d'un évènement crée d'autres évènements pour déplacer des segments obstrués, ces segments ayant été placés avant le segment courant, leurs priorités devaient être supérieures et le sont toujours. Donc, lors de leur remise dans la queue, ils seront re-routés avant le segment courant (leur ordre relatif n'aura pas changé). Dans certains cas on souhaite altérer cet ordre, on utilise alors le niveau d'évènement.

5.3.4 Description simplifiée de la fonction de coût

Pour chaque évènement, c'est-à-dire chaque traitement de segment, la fonction de coût mesure la difficulté à insérer un segment dans une piste donnée. Le coût d'insertion dans une piste dépend des segments déjà insérés dans la piste. En conséquence, tant qu'un segment n'est pas inséré dans une piste de routage, il n'est pas pris en compte. La fonction compare dans, l'ordre, les éléments suivants (Fig. 5.19) :

1. **Critère n°1 : Un drapeau *infini***, le segment recouvre un obstacle ou une partie fixe d'un net, il est impossible de le placer sur cette piste.
2. **Critère n°2 : L'écart par rapport au *rip-up* count de la piste cible** : En l'absence de ce critère, les segments, en cherchant la position optimale, vont avoir tendance à se concentrer sur certaines pistes et à se *riper* réciproquement pour l'obtenir. Il en résulte une augmentation trop rapide du *rip-up* count. Pour contrer cet effet, on introduit le critère suivant qui va étaler les segments sur les pistes à plus faible *rip-up*. La différence des *rip-up* count est prise en compte si l'écart est supérieur à 3 (paramètre ajustable). Les portions de pistes n'ont pas de *rip-up count* en propre. C'est le maximum des *rip-up counts* des segments occupant la portion de piste.
3. **Critère n°3 : Le plus petit delta**, le delta est la longueur en superposition avec un autre net dans la piste, diminuée de la longueur en superposition avec d'autres segments appartenant au même net qui seraient déjà insérés dans la piste (favorise l'alignement de segments du même net). La piste est totalement libre quand le delta est nul ou négatif.
4. **Critère n°4 : La distance à la position demandée de l'axe**, cette position est initialement la position optimale, mais au cours de l'exécution du programme elle peut être changée. Par exemple, on peut vouloir placer les perpendiculaires à un segment de façon particulière.

5. **Critère n°5 : L'allongement de la distance perpendiculaire**, c'est-à-dire la somme des distances aux attracteurs perpendiculaires. On cherche à se rapprocher de la position optimale.
6. **Critère n°6 : Pour départager les ex-aequo et garantir le déterminisme**, la valeur de la position de l'axe.

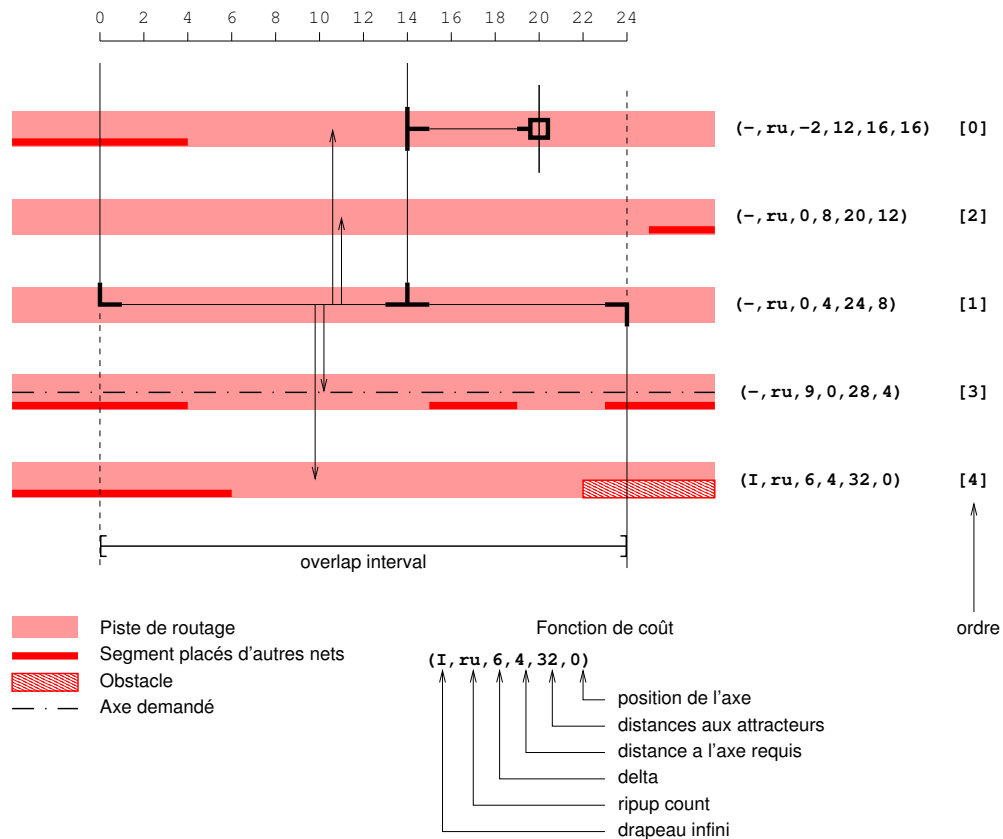


FIGURE 5.19 – Fonction de coût

5.3.5 Automate d'état d'assouplissement

Les évènements sont dépilés un à un et deux cas se présentent :

- Soit on trouve une piste libre dans laquelle placer le segment, le segment y est placé et son traitement s'arrête là.
- Soit il existe des conflits sur toutes les pistes candidates. Dans ce cas, on analyse le conflit dans la meilleure piste candidate, on en déduit des requêtes sur d'autres segments et on remet l'évènement du segment dans la pile.

Un segment peut être remis dans la pile, soit parce qu'aucune piste libre n'est disponible, soit parce qu'il fait l'objet d'une requête de *rip-up* par un autre segment. Dans tous les cas, le compteur de *rip-up* est incrémenté à chaque fois qu'un évènement est repoussé dans la pile.

Les topologies des signaux étant initialement assez rigides, le *rip-up and re-route* n'est pas suffisant pour assurer à lui seul un placement des segments sans recouvrement. Un

mécanisme d'assouplissement a donc été ajouté.

À chaque segment est associé un automate d'état qui gère son assouplissement progressif. **Tous les 5 essais d'insertion**, le segment passe par une étape d'assouplissement. La dernière étape étant l'échec du routage, le segment ne peut être placé.

Les états d'un automate d'un segment sont les suivants :

1. État initial.
2. **Rip-up** des perpendiculaires. Le placement des perpendiculaires peut bloquer le placement, on les dé-route puis on force le routage du segment courant en augmentant son niveau d'évènement.
3. **Minimisation** : Toujours en jouant sur la position des perpendiculaires, on cherche à minimiser la longueur du segment.
4. **Dogleg** : On casse le segment en deux morceaux.
5. **Résolution par analyse de l'historique** : En utilisant l'historique des évènements de routage, on casse à nouveau le segment.
6. **Changement de layer** : Dans les technologies où cela est possible, on remplace le layer du segment par un layer de niveau supérieur.
7. **Échec du routage** : Le segment ne sera pas placé et le routage détaillé du circuit a échoué.

Exemple détaillé du *rip-up* des perpendiculaires

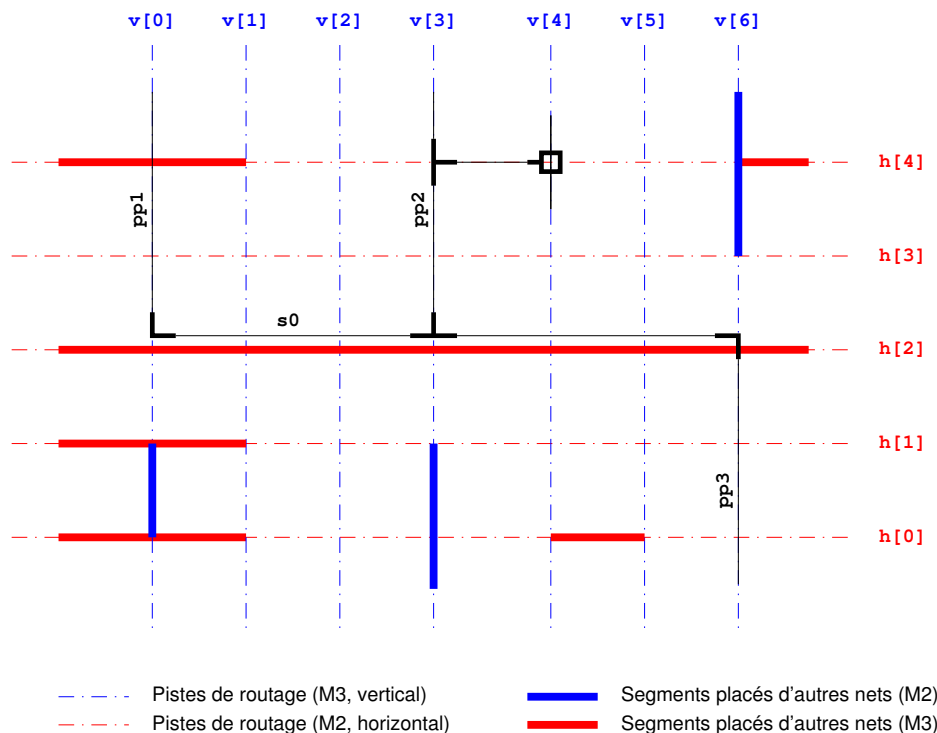


FIGURE 5.20 – *Rip-up* des perpendiculaires

La figure 5.20 détaille la tentative de placement du segment s_0 . L'ordonnancement a fait que ses trois perpendiculaires pp_1 , pp_2 et pp_3 ont été placées avant lui. Compte tenu

des autres segments placés dans les pistes perpendiculaires $v[0]$, $v[3]$ et $v[6]$, la seule piste sur laquelle on peut placer $s0$ est $h[2]$, mais celle-ci est déjà totalement prise. L'algorithme va tenter d'y placer $s0$ cinq de fois, puis passer à une étape d'assouplissement. Dans notre cas on supposera qu'il s'agit de l'étape de *rip-up* des perpendiculaires.

Nous allons alors retirer de leurs pistes (riper) les trois segments perpendiculaires puis les re-router. Si l'on remet directement $pp1$, $pp2$, $pp3$ et $s0$ dans la pile de priorité, ils vont être replacés dans le même ordre que celui qui a amené ce problème, soit :

```
Event queue :
[ 12] event( 0, 25, pp1 )
[ 13] event( 0, 31, pp2 )
[ 14] event( 0, 64, pp3 )
[ 15] event( 0, 102, s0 )
```

Il faut que $s0$ soit remis dans la pile de priorité avant ses perpendiculaires, on augmente donc sa priorité, pour avoir dans la pile :

```
Event queue :
[ 12] event( 1, 102, s0 )
[ 13] event( 0, 25, pp1 )
[ 14] event( 0, 31, pp2 )
[ 15] event( 0, 64, pp3 )
```

$s0$ étant correctement placé dans $h[3]$, lorsque $pp3$ va se replacer, il va utiliser la piste $v[5]$. Le résultat est illustré 5.21.

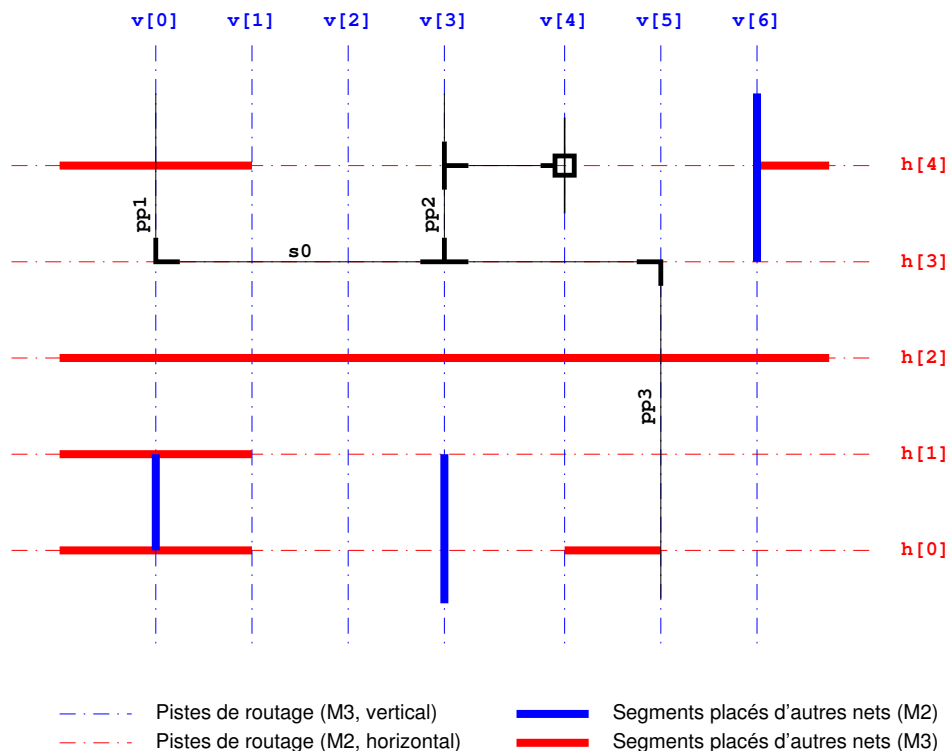


FIGURE 5.21 – Suite au *rip-up* des perpendiculaires

Détail de la minimisation de la longueur de fil

La minimisation de la longueur de fil fonctionne de façon similaire au *rip-up* des perpendiculaires. En plus de dé-router les perpendiculaires, on va rechercher dans la piste $h[3]$ un espace libre où l'on pourrait placer s_0 si celui-ci était réduit après déplacement des perpendiculaires. Cette fois-ci, on ne change pas l'ordre de la pile, les perpendiculaires seront placées avant s_0 , mais on demande à $pp1$, $pp2$ et $pp3$ d'essayer de se rapprocher le plus possible de la piste $v[2]$. Un résultat possible est illustré figure 5.22.

On peut, à l'occasion, remarquer comment évolue la topologie des fils. Les deux branches du HTee (intersection $h[3]$ et $v[3]$) vont apparaître visuellement comme un coude car ses deux segments partent vers la gauche, et les deux coudes vers $pp1$ et $pp2$ s'alignent et apparaîtront comme un VTee. Ce placement des segments n'en est pas moins valide.

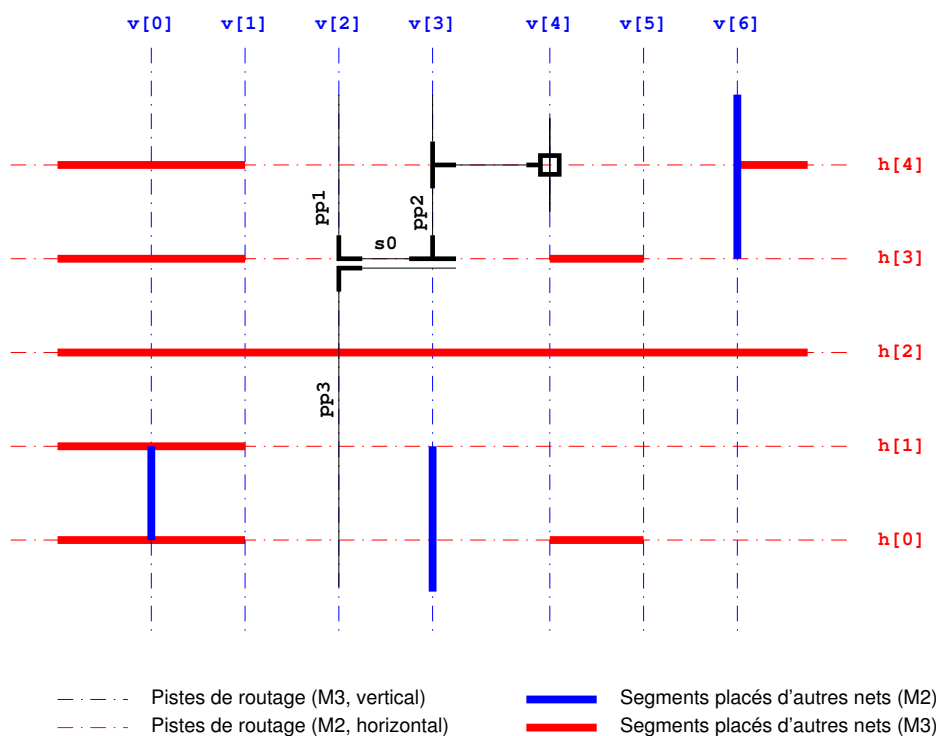


FIGURE 5.22 – Résultat de la minimisation

5.4 Contraintes des signaux analogiques

La structure de données des segments articulés, avec pour seules opérations de modifications le déplacement d'un segment sur son axe ou la création d'un double *dogleg*, permettent d'assurer un contrôle très précis de la forme des *nets*.

5.4.1 Routage symétrique

Par principe, pour construire un routage détaillé symétrique il est obligatoire que le routage global soit lui même symétrique. À partir d'un routage global symétrique entre *net A* et *net B*, la construction du routage détaillé étant déterministe, nous allons coupler

(un pour chaque segment). La figure 5.24 montre comment les pistes sont appariées pour assurer la symétrie, dans le cas de segments parallèles à l'axe de symétrie (*id* : 54 et *id* : 76) et de segments perpendiculaires à l'axe de symétrie (*id* : 52 et *id* : 78). La fonction de coût d'insertion est la somme des deux coûts pour chaque segment (ou logique dans le cas des drapeaux booléens).

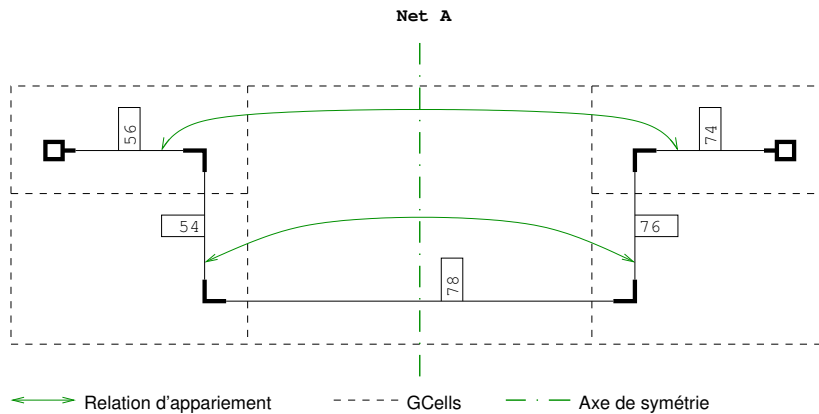


FIGURE 5.25 – Exemple d'un *net* symétrique à lui-même

Un deuxième cas de symétrie est possible, un *net* par rapport à lui-même selon un axe de symétrie centrale (cf. figure 5.25). Dans ce cas, nous aurons des évènements couplant deux segments du même *net* et dans le cas de l'unique segment traversant l'axe de symétrie, nous aurons un évènement à un seul segment.

Event queue :

```
[ 12] event( 0, 56, netA_id:54, netA_id:76 )
[ 13] event( 0, 67, netA_id:56, netA_id:74 )
[ 14] event( 0, 164, netA_id:78 )
```

Segment central : L'algorithme de routage global est conçu pour ne fabriquer que des arbres (pas de circuits fermés), il ne peut y avoir qu'un seul segment traversant l'axe de symétrie (démonstration par l'absurde, si on en a deux, on peut construire une boucle ce qui est contradictoire avec l'hypothèse).

5.4.2 Segments larges

La structure de données des segments permet aussi de gérer des segments dont la largeur tient sur plusieurs pistes. Si les segments eux-mêmes peuvent avoir une largeur quelconque, nous procédons à une discrétisation au niveau de la tuile qui va représenter leur empattement dans la structure de données. Les tuiles doivent avoir une largeur équivalente à un nombre entier de pistes. La figure 5.26 montre une tuile pour un segment dont la largeur est plus grande qu'une piste, mais moins que deux. Sa tuile occupera donc deux pistes.

Concernant le calcul de la fonction de coût d'insertion, nous nous trouvons en fait dans une situation identique à celle rencontrée en section 5.4.1, c'est à dire que le coût est la somme du coût de plusieurs pistes individuelles. On utilise la même logique d'addition.

Isolation des signaux sensibles : Les segments larges introduisent en fait une fonctionnalité plus générale, la capacité de découpler la largeur réelle d'un segment de la place qu'il va occuper dans les pistes de routages. Ceci est particulièrement intéressant dans le cas des signaux sensibles au bruit et qui doivent être isolés latéralement de tout agresseur, il suffit de les déclarer plus larges qu'ils ne sont et le routeur les isolera par construction.

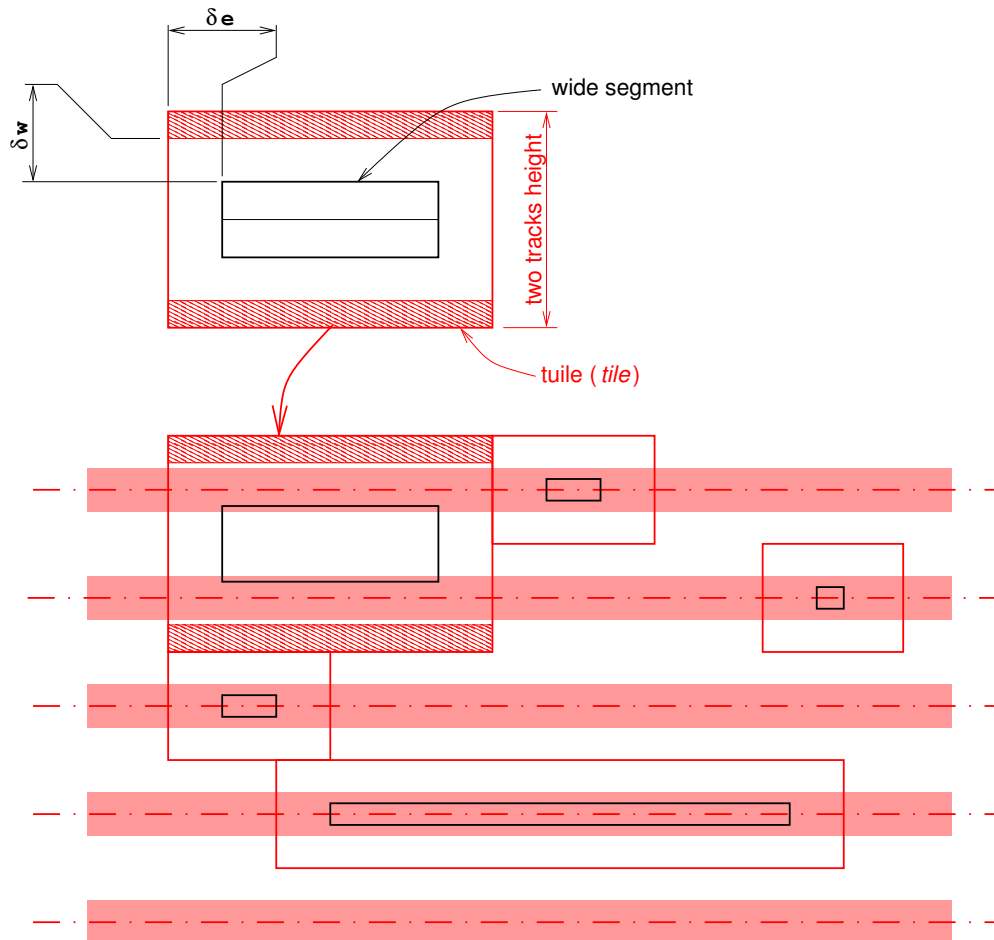


FIGURE 5.26 – Prise en compte des segments larges

5.5 Conclusion

Le routage détaillé correspond à l'étape finale de la génération du dessin des masques. En suivant notre approche de placement et routage, cette étape consiste à résoudre les superpositions de fils tout en préservant le passage à travers les chemins de *GCells* obtenus par l'étape du routage global. Pour cela, le résultat du routage global est transformé en un ensemble de briques de construction composé de segments et de contacts dont la position est contrainte par la *GCell* dans laquelle ils se trouvent. Le routeur cherche ensuite à positionner les fils des signaux à leur position optimale réduisant au mieux le nombre de contacts ainsi que la longueur de fils totale. Cette topologie initiale, souvent impossible à placer sans superposition de fils, peut subir alors un certain nombre d'étapes d'assouplissement telles que la création de *doglegs* ou changement de *layers*. Ces étapes d'assouplissement sont conditionnées par des automates propres à chaque segment. Le routeur détaillé gère également des contraintes spécifiques analogiques, en particulier les signaux symétriques en gérant la superposition des fils de manière couplée.

Initialement, le routeur détaillé a été développé par Jean-Paul Chaput dont l'application, avant cette thèse, était exclusivement destinée aux circuits numériques. Un travail de coordination avec Jean-Paul Chaput a été réalisé pour intégrer des parties analogiques et mixtes incluant la gestion topologique (organisation des *GCells* non matricielles) et de contraintes supplémentaires (routage détaillé symétrique). La contribution des travaux de thèse au routage détaillé se limite à la transformation du routage global en briques de construction du routeur détaillé. Il nous a semblé pertinent de présenter le fonctionnement général du routeur détaillé afin de présenter les intérêts qu'ont suscités ce sujet de thèse et de montrer l'adéquation entre routeur global et routeur détaillé.

5.6 Références

- [1] Koichi Mikami. A computer program for optimal routing of printed circuit connections. In *Proc. IFIPs*, volume 47, pages 1475–1478, 1968. [130](#)
- [2] David W Hightower. A solution to line routing problems on the continuous plane. In *Papers on Twenty-five years of electronic design automation*, pages 11–34. ACM, 1988. [130](#)

Chapitre 6

Résultats

Sommaire

6.1 Introduction	154
6.2 Amplificateur à transconductance de type Miller	155
6.3 Amplificateur à source de courant ajustable	162
6.4 Transconductance différentielle configurable	166
6.5 Conclusion	169
6.6 Références	169

6.1 Introduction

Pour illustrer notre méthodologie de placement routage de circuits mixtes, nous utilisons notre approche de placement et routage décrite dans cette thèse pour concevoir trois circuits de différentes tailles : un amplificateur à transconductance, un amplificateur à source de courant ajustable et une transconductance différentielle contrôlable. On rappelle qu'en entrée de la phase de placement-routage de notre approche, il est nécessaire de connaître la *netlist*, le dimensionnement des transistors du circuit et les parties numériques préalablement placées.

Le Laboratoire d'Informatique de Paris 6 a développé au cours des vingt dernières années des outils de CAO dédiés à la conception de circuits intégrés. La figure 6.1 présente le flot de conception de circuits analogiques et mixtes utilisé pour la réalisation des trois circuits :

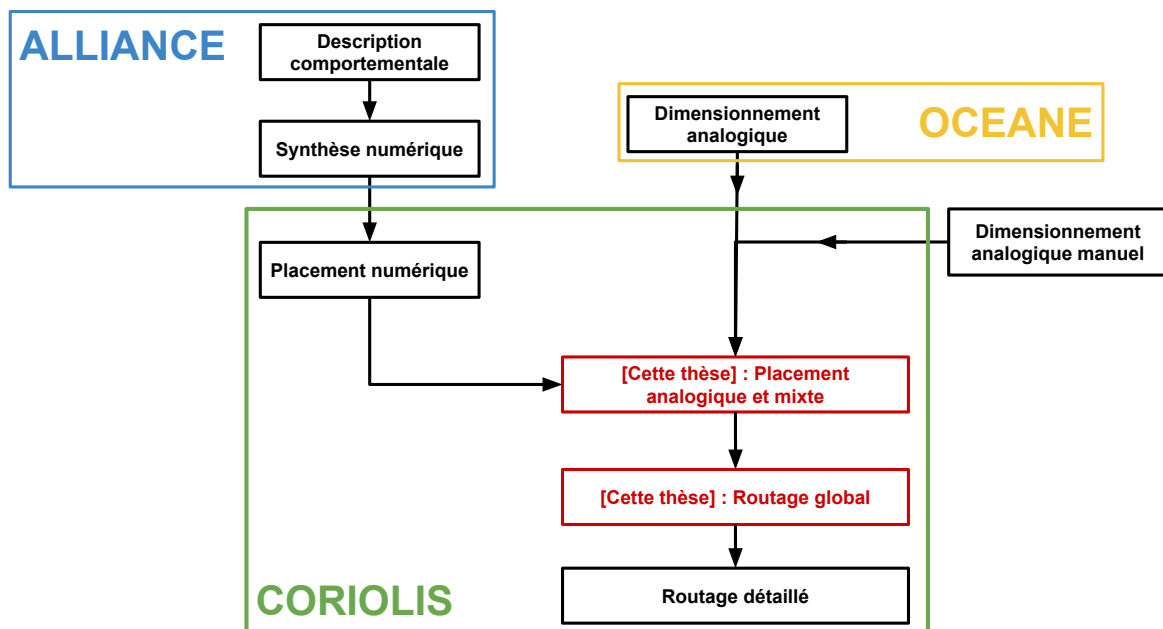


FIGURE 6.1 – Flot de conception mixte à travers les outils logiciels du LIP6

Les étapes allant de la description comportementale à la synthèse logique sont réalisées à partir de la chaîne d'outils d'**Alliance**[1] (en bleu sur la figure 6.1). Les différentes étapes de la synthèse logique sont réalisées à partir des outils :

- **SYF** : Synthèse d'un automate en un réseau booléen.
- **BOOM** : Optimisation d'un réseau booléen.
- **BOOG** : Transformation en une *netlist* de cellules précaractérisées.
- **LOON** : Optimisation électrique locale.

Concernant le dimensionnement du circuit analogique, **OCEANE**[2] (en jaune sur la figure 6.1) est conçu pour recevoir en entrée un jeu de spécifications de haut niveau portant sur un bloc fonctionnel ou une fonction élémentaire analogique. **OCEANE** produit une *netlist* au format standard *SPICE*, correspondant à une description topologique des composants élémentaires (transistors, résistances, condensateurs et inductances) complètement dimensionnés et affecte leurs paramètres de style pour la génération des masques

correspondante. Un interfaçage avec d'autres outils éventuellement commerciaux (simulateurs, générateurs automatiques de masques) est ainsi directement réalisable. Le concepteur peut également choisir de dimensionner manuellement ou à travers un autre outil son circuit analogique.

Le placeur numérique EtesianG ainsi que l'outil de placement analogique et mixte, et le routeur global et détaillé font partie de **Coriolis**[3]. La méthodologie complète du placement routage analogique et mixte décrite dans cette thèse est appliquée aux circuits considérés. Pour rappel, la méthodologie de placement routage suit les étapes suivantes :

1. **[Placement analogique et mixte] Étape 1** : Instanciation des modules analogiques et numériques du circuit (fonction atomique analogique, dimensions, ...).
2. **[Placement analogique et mixte] Étape 2** : Description de la *netlist* (connexions entre les connecteurs du circuit et tailles des fils de routage pour chaque *net*).
3. **[Placement analogique et mixte] Étape 3** : Description du *slicing tree* et des contraintes considérées.
4. **[Placement analogique et mixte] Étape 4** : Exécution de l'algorithme de placement et choix du placement désiré.
5. **[Routage global] Étape 5** : Exécution de l'algorithme de routage global.
6. **[Routage global] Étape 6** : Exécution de l'algorithme de routage détaillé.

6.2 Amplificateur à transconductance de type Miller

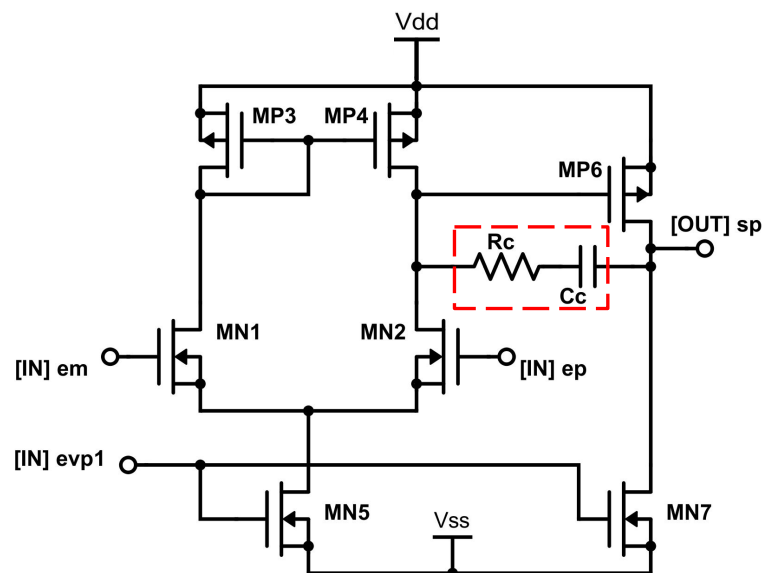


FIGURE 6.2 – Schéma électrique de l'amplificateur Miller à transconductance

Le premier circuit utilisé est un amplificateur à transconductance de type Miller, réalisé en technologie **130nm CMOS** et son schéma est illustré par la figure 6.2. Ce circuit est composé de 5 modules (1 paire différentielle, 1 miroir de courant et 3 transistors). Dans l'état actuel de **Coriolis**, seuls les modules à base de transistors sont implémentés (en particulier ceux listés dans le chapitre 3). C'est pourquoi le dessin des masques de la résistance R_c et de la capacité C_c ne peuvent être générés.

Pour cet exemple, nous proposons le placement présenté sur la figure 6.3.(a). Le *slicing tree* relatif à ce placement est représenté par la figure 6.3.(b).

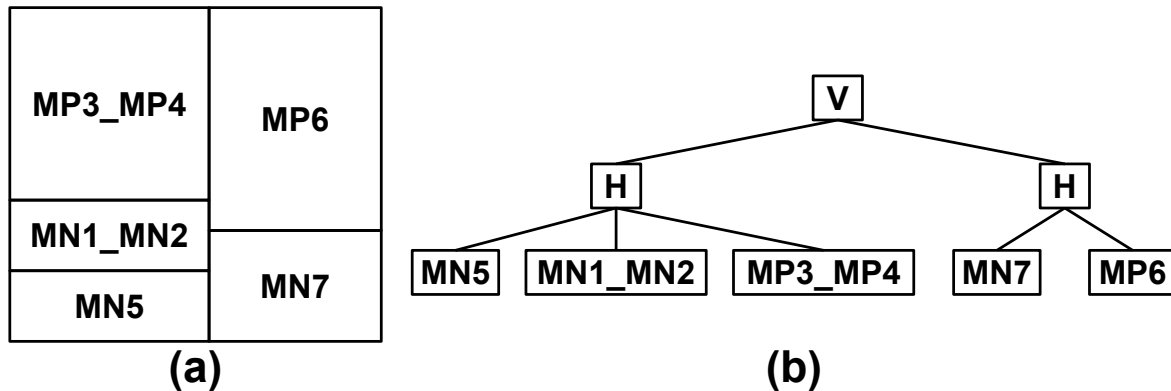


FIGURE 6.3 – Placement et *slicing tree* correspondant de l’amplificateur Miller

On propose de présenter l’efficacité de notre flot de conception présenté par la figure 6.1 en proposant deux versions de ce circuit avec des spécifications de dimensionnement différentes obtenu par l’outil **OCEANE**. Le dimensionnement des différents modules est le suivant :

Dimensionnement 1 :

- mp3_mp4 : $W = 13,74 \mu m$, $L = 0,45 \mu m$
- mn1_mn2 : $W = 4,99 \mu m$, $L = 0,45 \mu m$
- mn5 : $W = 12,02 \mu m$, $L = 0,45 \mu m$
- mp6 : $W = 68,7 \mu m$, $L = 0,45 \mu m$
- mn7 : $W = 23,44 \mu m$, $L = 0,45 \mu m$

Dimensionnement 2 :

- mp3_mp4 : $W = 78,84 \mu m$, $L = 0,46 \mu m$
- mn1_mn2 : $W = 27,86 \mu m$, $L = 0,46 \mu m$
- mn5 : $W = 67,82 \mu m$, $L = 0,46 \mu m$
- mp6 : $W = 399,2 \mu m$, $L = 0,46 \mu m$
- mn7 : $W = 141,12 \mu m$, $L = 0,46 \mu m$

La description du script *Python* de ce circuit est décrite par le code 6.1. On peut y observer la description des modules analogiques comprenant leurs noms, leurs dimensions et la description des *nets* du circuit. À titre d’exemple, on impose que les fils de routage du *net vout+* soient de tailles $0.6 \mu m$.

Code 6.1 – Script *Python* de l’amplificateur à transconductance - Description des modules et des *nets*

```

1 self.devicesSpecs = \
2 # | Class      | Instance | Layout Style | Type| W  | L  |Dum|SFirst|Bulk| BulkC |
3 # +-----+-----+-----+-----+-----+-----+-----+-----+-----+
4 [ [CurrentMirror , 'mp3_mp4', 'WIP CM'      , PMOS, 13.7, 0.45, 0, True, 0xf, True ]
5 , [DifferentialPair, 'mn1_mn2', 'WIP DP'      , NMOS, 4.99, 0.45, 0, True, 0xf, False ]
6 , [Transistor    , 'mn5'    , 'WIP Transistor', NMOS, 12.0, 0.45, 0, True, 0xf, True ]
7 , [Transistor    , 'mp6'    , 'WIP Transistor', PMOS, 68.7, 0.45, 0, True, 0xf, True ]
8 , [Transistor    , 'mn7'    , 'WIP Transistor', NMOS, 23.4, 0.45, 0, True, 0xf, True ]
9 ]
10
11 self.netTypes = \
12 # | Net      | Type      |
13 # +-----+-----+
14 { 'vin+' : { 'isExternal' : True }
15 , 'vin-' : { 'isExternal' : True }
16 , 'vout+' : { 'isExternal' : True }
17 , 'vbias' : { 'isExternal' : True }
18 , 'vdd' : { 'isExternal' : True }
19 , 'vss' : { 'isExternal' : True }

```

```

20 }
21
22 self.netSpecs = \
23 # | Net | Connector |
24 # +-----+-----+
25 { 'vin-' : [( 'mn1_mn2', 'G1' ), ]
26 , 'vin+' : [( 'mn1_mn2', 'G2' ), ]
27 , 'va' : [( 'mn1_mn2', 'D1' ), ( 'mp3_mp4', 'D1' ), ( 'mp3_mp4', 'G' )]
28 , 'vb' : [( 'mn1_mn2', 'D2' ), ( 'mp3_mp4', 'D2' ), ( 'mp6', 'G' )]
29 , 'vout+' : [{ 'W': '0.6' }, ( 'mp6', 'D' ), ( 'mn7', 'D' )]
30 , 'vz' : [( 'mn5', 'D' ), ( 'mn1_mn2', 'S' )]
31 , 'vbias' : [( 'mn5', 'G' ), ( 'mn7', 'G' )]
32 , 'vdd' : [( 'mp3_mp4', 'S' ), ( 'mp6', 'S' )]
33 , 'vss' : [( 'mn1_mn2', 'B' ), ( 'mn5', 'S' ), ( 'mn7', 'S' )]
34 }
35
36 self.readParameters( './oceane-results.txt' )

```

Dans notre cas, nous chargeons les dimensions des transistors à partir d'un fichier généré par **OCEANE** à partir de la méthode `readParameters(file)` (voir partie 3.5.1). Les valeurs des W et L indiquées dans le code 6.1 ne sont donc pas utilisées.

La description du *slicing tree* de la figure 6.3 est présenté par le code 6.2. Le nombre de facteurs de forme possibles pour chacun des modules est le suivant (ces facteurs de forme restent les mêmes pour les deux versions du circuit) :

- Pour $mn5$: quatre facteurs de forme.
- Pour $mn1_mn2$: deux facteurs de forme.
- Pour $mp3_mp4$: deux facteurs de forme.
- Pour $mn7$: dix facteurs de forme.
- Pour $mp6$: dix facteurs de forme.

On utilise également le paramètre de tolérance qu'on fixe à $5\mu m$ dans les nœuds hiérarchique horizontaux et à $20\mu m$ dans les nœuds hiérarchiques verticaux.

Code 6.2 – Script *Python* de l'amplificateur à transconductance - Description du *slicing tree*

```

1 self.beginCell( 'millerOta' )
2 self.doDevices()
3 self.doNets ()
4
5 self.beginSlicingTree ()
6 self.setToleranceBandH ( 5 )
7 self.setToleranceBandW ( 20 )
8 self.pushVNode( Center )
9
10 self.pushHNode( Center )
11 self.addDevice( 'mn5', Center, span=(1.0, 1.0, 4.0) )
12 self.addDevice( 'mn1_mn2', Center, span=(2.0, 2.0, 2.0) )
13 self.addDevice( 'mp3_mp4', Center, span=(2.0, 2.0, 2.0) )
14 self.popNode()
15
16 self.pushHNode( Center )
17 self.addDevice( 'mn7', Center, span=(2.0, 1.0, 10.0) )
18 self.addDevice( 'mp6', Center, span=(2.0, 1.0, 10.0) )
19 self.popNode()
20
21 self.popNode()
22 self.endSlicingTree ()
23 self.endCell ()

```

Suite à l'exécution du script *Python*, l'algorithme de placement calcule l'ensemble des facteurs de forme possible.

Pour le dimensionnement 1

Une fois le nombre de facteurs de forme déterminé pour le circuit et avec le dimensionnement 1, ces résultats sont présentés sous la forme d'un graphe de points illustré par la figure 6.4.

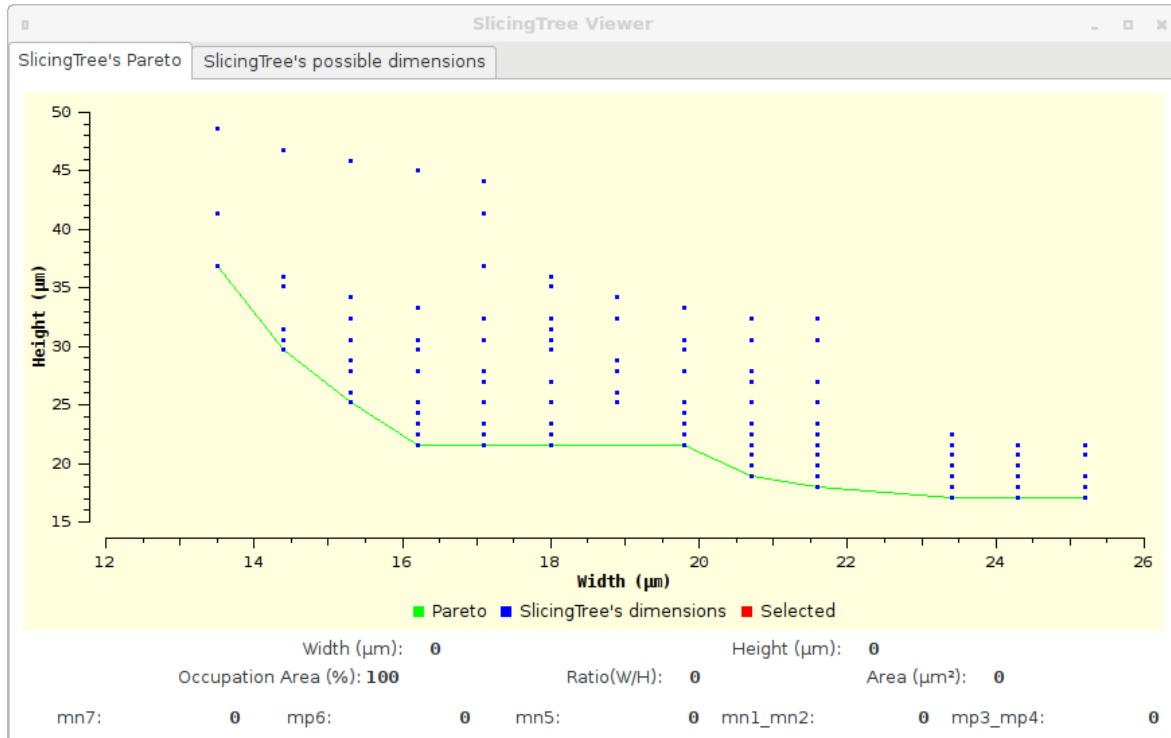


FIGURE 6.4 – Pour le dimensionnement 1 - Graphe de points des placements possibles

Quelques informations associées aux résultats de placements sont présentées ci-dessous :

- **Nombre total de placements valides** : 90 placements différents.
- **Placement occupant le moins de surface** : $349,92 \mu m^2$.
- **Placement occupant le plus de surface** : $754,11 \mu m^2$.
- **Temps d'exécution du placement** : 0,184 secondes.

En prenant en compte les paramètres de tolérance, l'algorithme élimine 47 placements pour conserver 90 placements différents valides.

En choisissant le placement occupant le moins de surface à partir du graphe de points, on exécute la phase de placement de cette solution et le routage global. Le résultat est présenté sur la figure 6.5. On peut y observer en rouge les contours des *GCells* destinées à la recherche des arbres d'interconnexions de la phase de routage global, chaque rectangle rouge définit une *GCell* représentant soit un espace/canal de routage soit un module.

Après analyse du résultat obtenu, le concepteur est libre de choisir une solution différente sur le graphe de points. Dans la mesure où cette solution lui semble correcte, on exécute la phase de routage détaillé et le résultat final est présenté par la figure 6.6.

Quelques informations à propos du routage global et du routage détaillé sont présentées ci-dessous :

6.2. AMPLIFICATEUR À TRANSCONDUCTANCE DE TYPE MILLER

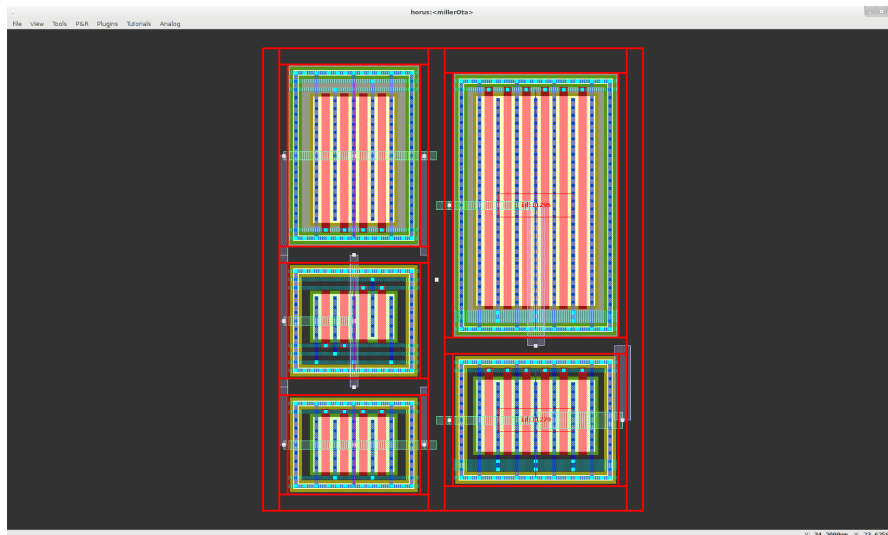


FIGURE 6.5 – Transistors de l’amplificateur Miller suite au routage global

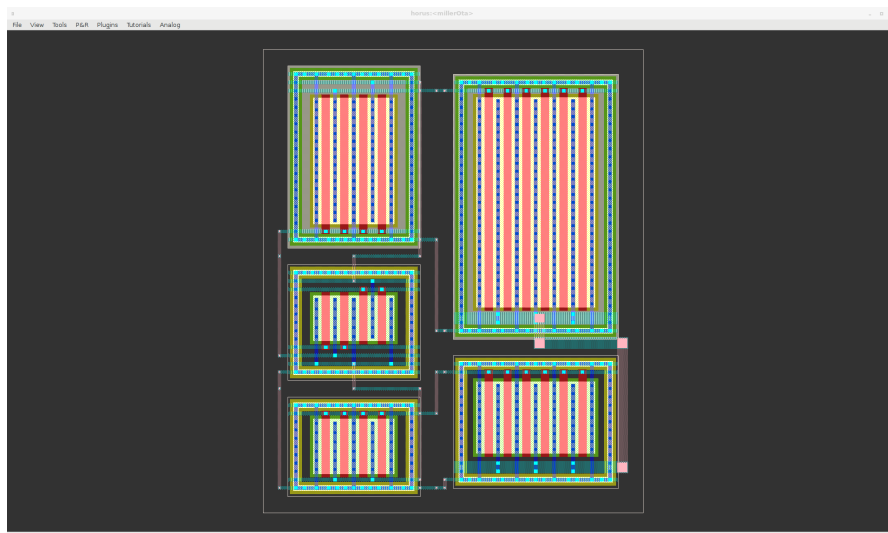


FIGURE 6.6 – Amplificateur Miller (sans la résistance et la capacité) placé et routé

- **Nombre de *GCells*** : 25 *GCells*.
- **Temps d’exécution du routage global** : 0,130 secondes.
- **Surface suite à l’étirement des canaux** : $521,64 \mu m^2$.
- **Longueur de fil** : $1468 \mu m$.
- **Nombre de segments** : 49 segments.
- **Nombre de contacts** : 57 contacts.
- **Temps d’exécution du routage détaillé** : 0,024 secondes.

Pour le dimensionnement 2

Une fois le nombre de facteurs de forme déterminé pour le circuit et avec le dimensionnement 2, ces résultats sont présentés sous la forme d'un graphe de points illustré par la figure 6.7.

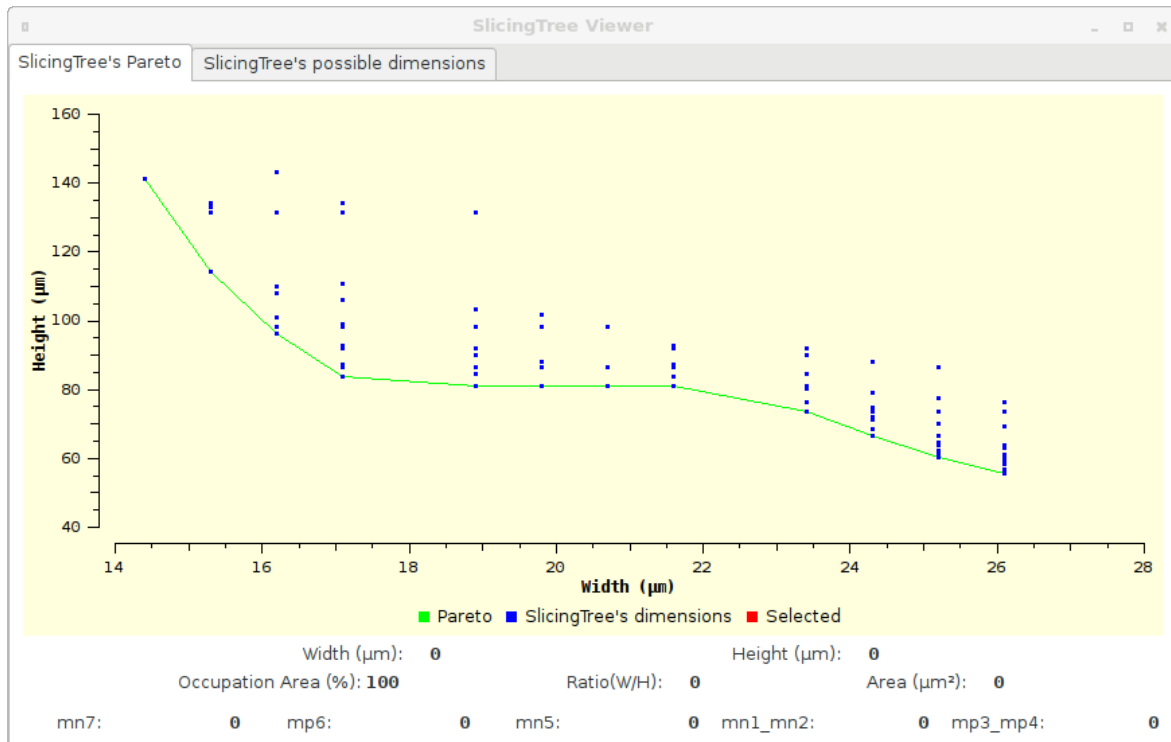


FIGURE 6.7 – Pour le dimensionnement 2 - Graphe de points des placements possibles

Quelques informations associées aux résultats de placements sont présentées ci-dessous :

- **Nombre total de placements valides** : 80 placements différents.
- **Placement occupant le moins de surface** : $1431,27 \mu m^2$.
- **Placement occupant le plus de surface** : $2483,46 \mu m^2$.
- **Temps d'exécution du placement** : 0.222 secondes.

En prenant en compte les paramètres de tolérance, l'algorithme élimine 141 placements pour conserver 80 placements différents valides.

En choisissant le placement occupant le moins de surface à partir du graphe de points, on exécute la phase de placement de cette solution et le routage global. Le résultat est présenté sur la figure 6.8. On peut y observer en rouge les contours des *GCells* destinées à la recherche des arbres d'interconnexions de la phase de routage global, chaque rectangle rouge définit une *GCell* représentant soit un espace/canal de routage soit un module.

Après analyse du résultat obtenu, le concepteur est libre de choisir une solution différente sur le graphe de points. Dans la mesure où cette solution lui semble correcte, on exécute la phase de routage détaillé et le résultat final est présenté par la figure 6.9.

Quelques informations à propos du routage global et du routage détaillé sont présentées ci-dessous :

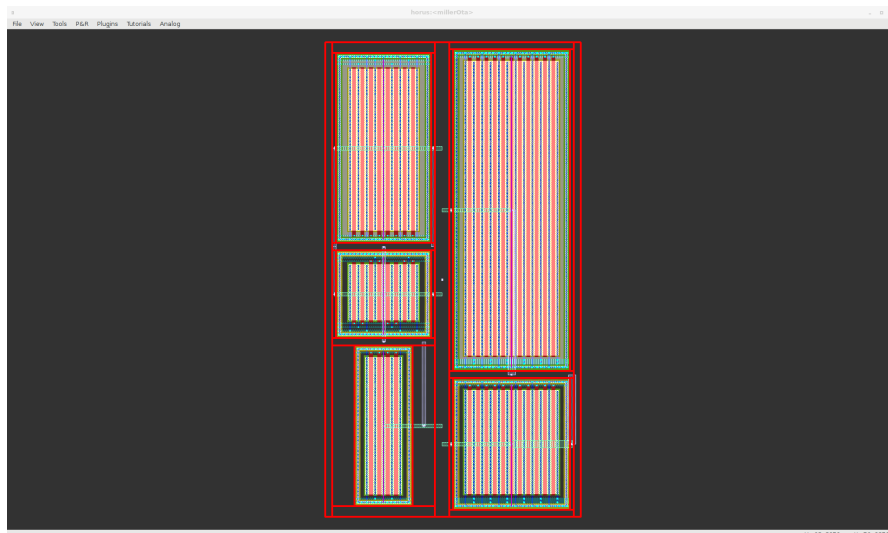


FIGURE 6.8 – Routage global des transistors de l'amplificateur Miller, dimensionnement 2

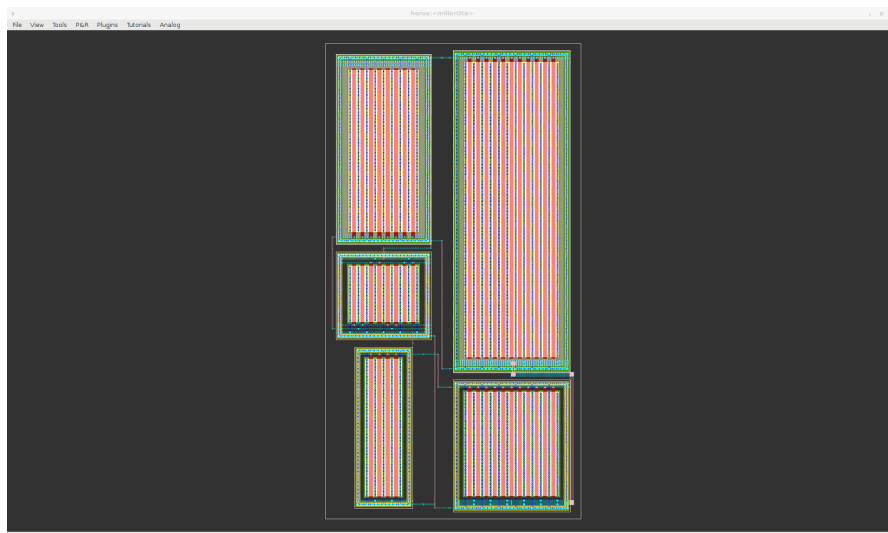


FIGURE 6.9 – Transistors de l'amplificateur Miller placés et routés, dimensionnement 2

- **Nombre de *GCells*** : 25 *GCells*.
- **Temps d'exécution du routage global** : 0,066 secondes.
- **Surface suite à l'étirement des canaux** : 1866,24 μm^2 .
- **Longueur de fil** : 2643 μm .
- **Nombre de segments** : 49 segments
- **Nombre de contacts** : 57 contacts.
- **Temps d'exécution du routage détaillé** : 0,020 secondes.

6.3 Amplificateur à source de courant ajustable

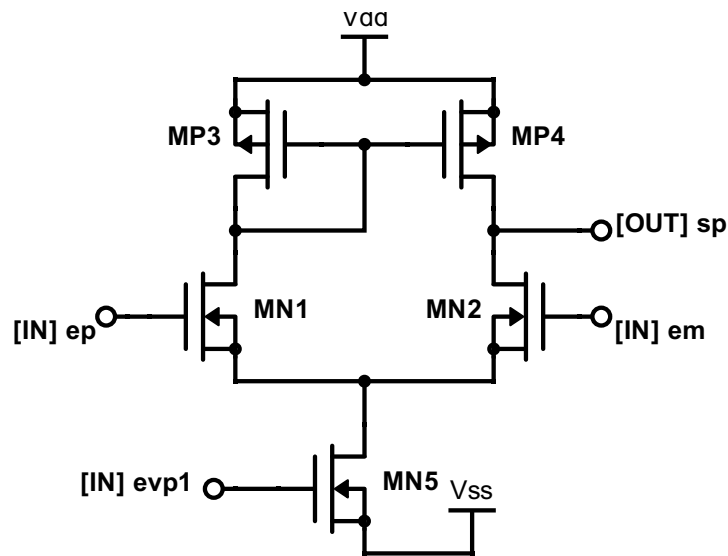


FIGURE 6.10 – Schéma électrique de l'amplificateur à source de courant ajustable

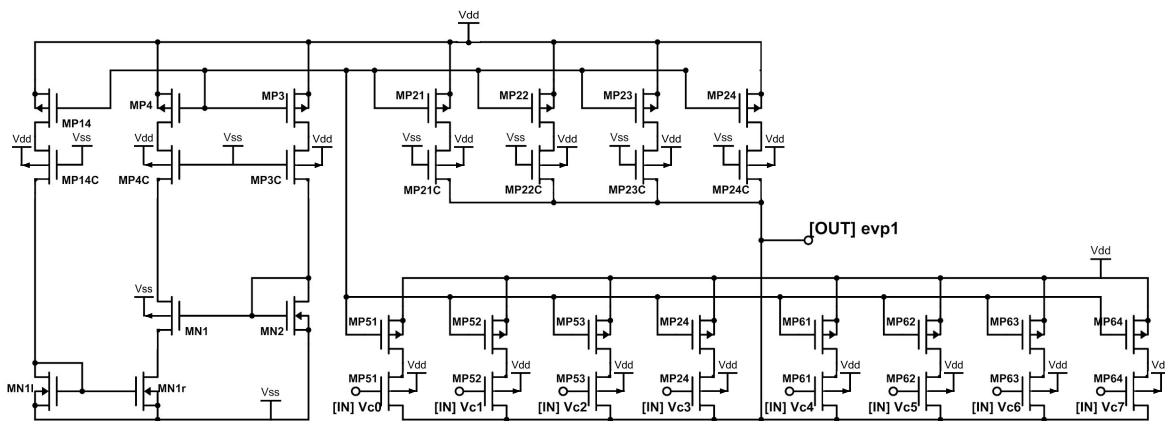


FIGURE 6.11 – Schéma électrique de la polarisation

Le second circuit utilisé, pour vérifier notre méthodologie de conception de circuit mixte, est un amplificateur à source de courant ajustable réalisé en technologie **350nm CMOS** et son schéma est illustré par la figure 6.10. Le circuit de polarisation de l'amplificateur (en entrée *evp1* sur la figure 6.10) est représenté par le schéma électrique illustré par la figure 6.11. 8 transistors du circuit de polarisation sont contrôlés par un décodeur numérique dont le schéma électrique est présenté par la figure 6.12. Ce décodeur est contrôlé par 4 bits d'entrée et retourne en sortie 8 bits selon les conditions de la figure 6.12. Ces 8 bits de sortie du décodeur sont les entrées vers les 8 transistors du circuit de polarisation.

Pour cet exemple, nous proposons le placement présenté sur la figure 6.13. Le *slicing tree* relatif à ce placement est représenté par la figure 6.14. La description du script *Python* de ce circuit est décrite par le code A.1 se trouvant en annexe. Le circuit complet est composé de 41 modules (40 transistors, un circuit numérique placé). Ce circuit comprend 2

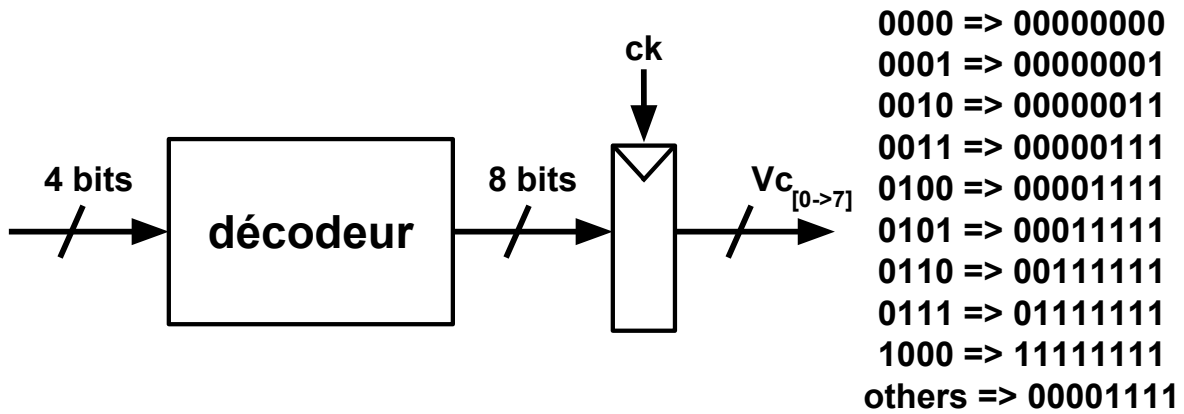


FIGURE 6.12 – Décodeur numérique contrôlant la polarisation

contraintes de placements symétriques.

Hrail										
MP14	MP4	MP3	MP21	MP22	MP23	MP24	MP10TA	MP20TA		
MPC22			MPC23			MPC24				
MP14C		MP4C		MP3C		MPC21			MN10TA	MN20TA
MP251	MP252	MP253	MP254	MP261	MP262	MP263	MP264			
MPC251	MPC252	MPC253	MPC254	MPC261	MPC262	MPC263	MPC264			
MN1r	MN2				MNP11			MN50TA		
MN1l										
MN1										
Hrail										

FIGURE 6.13 – Placement de l’amplificateur à source de courant ajustable

La description du *slicing tree* de la figure 6.14 est présenté par le code A.1. Le nombre de facteurs de forme pour chacun des modules est le suivant :

- Pour *mn2* : huit facteurs de forme.
- Pour *mp11*, *mn1l* et *mnr1* : deux facteurs de forme.
- Pour les 38 autres modules : un facteur de forme.

Suite à l’exécution du script *Python*, notre outil de placement calcule l’ensemble des facteurs de forme possible. Une fois le nombre de facteurs de forme déterminé pour le circuit, ces résultats sont présentés sous la forme d’un graphe de point illustré par la figure 6.15.

Quelques informations liées aux résultats de placements possibles sont présentées ci-dessous :

- **Nombre total de placements valides** : 48 placements.
- **Placement occupant le moins de surface** : 59535 μm^2 .

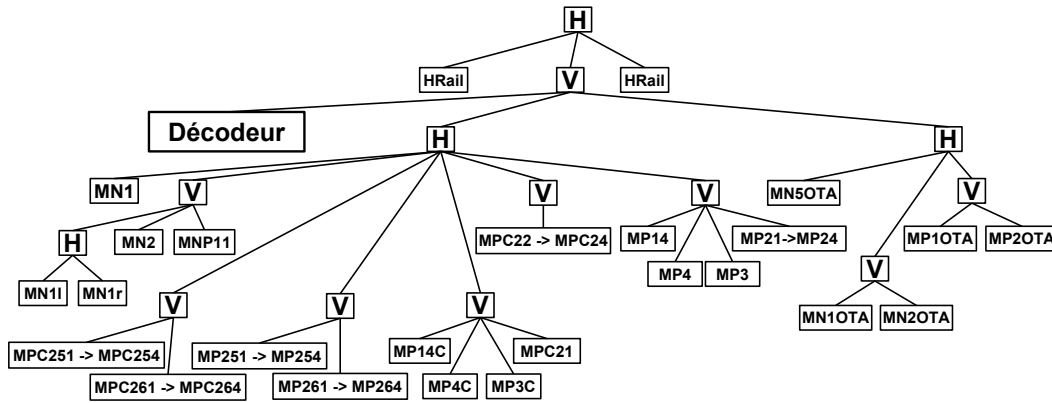


FIGURE 6.14 – Slicing tree du placement de l’amplificateur à source de courant ajustable

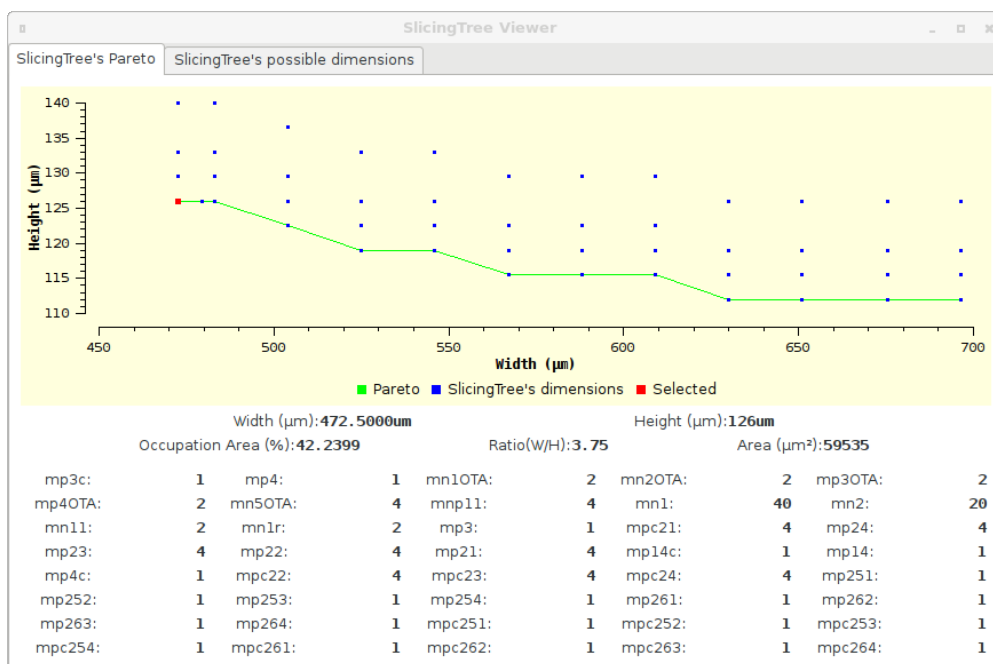


FIGURE 6.15 – Graphe de points des placements possibles

- **Placement occupant le plus de surface :** $87759 \mu m^2$.
- **Temps d’exécution du placement :** 1,680 secondes.

En tant qu’exemple, on choisit le circuit de plus petite surface auquel on applique le routage global et le routage détaillé. Le résultat obtenu est présenté par la figure 6.16.

Quelques informations à propos du routage global et du routage détaillé sont présentées ci-dessous :

- **Nombre de GCells :** 258 GCells.
- **Temps d’exécution du routage global :** 0,743 secondes.
- **Surface suite à l’étirement des canaux :** $110580.75 \mu m^2$.
- **Longueur de fil :** $37384 \mu m$.
- **Nombre de segments et de contacts :** 1228 segments et 1428 contacts
- **Temps d’exécution du routage détaillé :** 1,025 secondes.

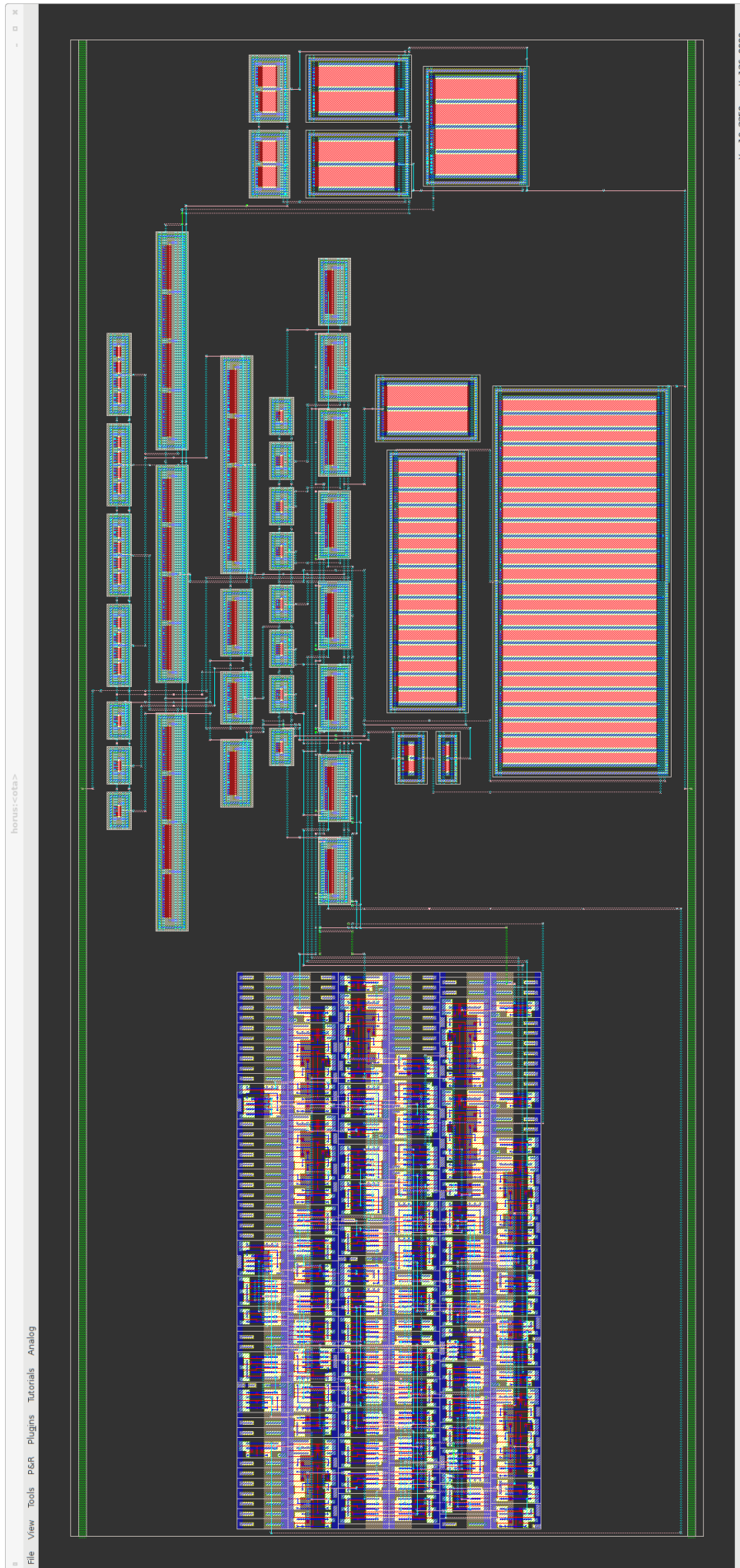


FIGURE 6.16 – Amplificateur à source de courant ajustable placé et routé

6.4 Transconductance différentielle configurable

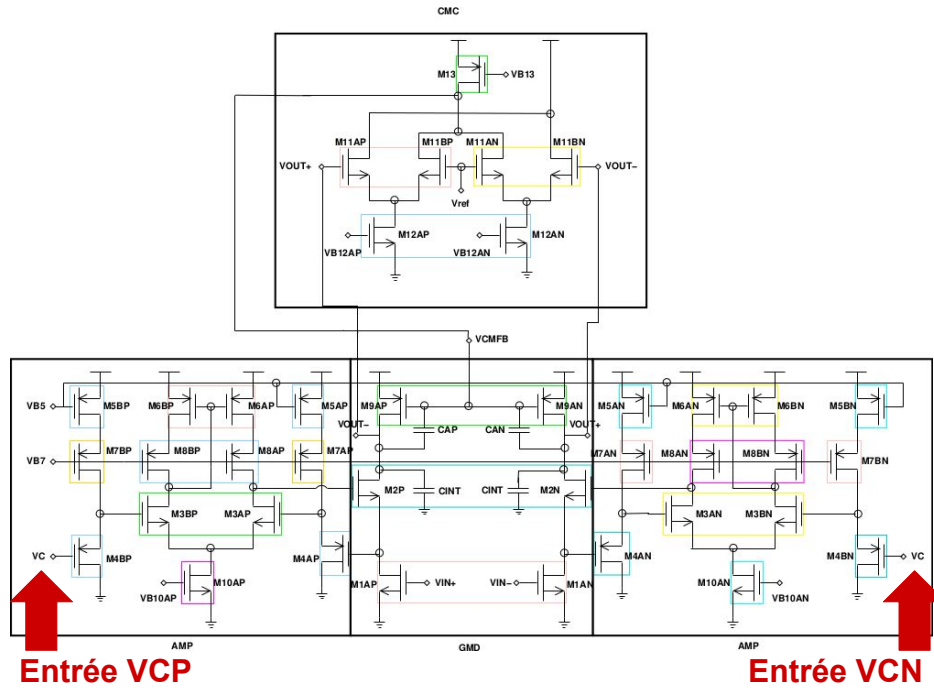


FIGURE 6.17 – Schéma électrique de la transconductance différentielle contrôlable

Le troisième circuit utilisé pour vérifier notre méthodologie de conception de circuit mixte est une transconductance différentielle[4] configurable numériquement, réalisée en technologie **130nm CMOS** et son schéma est illustré par la figure 6.17. Ce circuit est configuré par les entrées différentielles *VCP* et *VCN* dont le contrôle est obtenu par la sortie du circuit de la figure 6.18. Les entrées *VC1* à *VC7* correspondent aux sortie du décodeur de la figure 6.19.

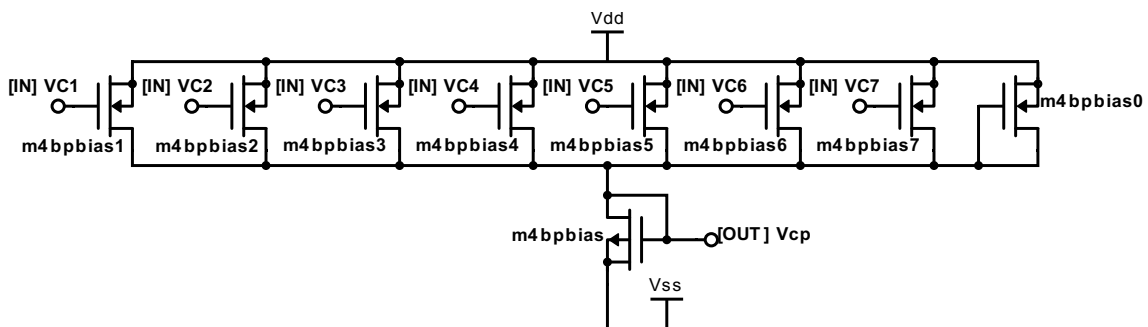


FIGURE 6.18 – Circuit intermédiaire entre les sorties du décodeur et de l'entrée VC

Le circuit complet est composé de 49 modules (6 paires différentielle, 3 montages source commune, un circuit numérique (décodeur) et 39 transistors). Ce circuit comprend 9 contraintes de placements symétriques, 9 paires de *nets* symétriques et 2 *nets* de symétrie unique.

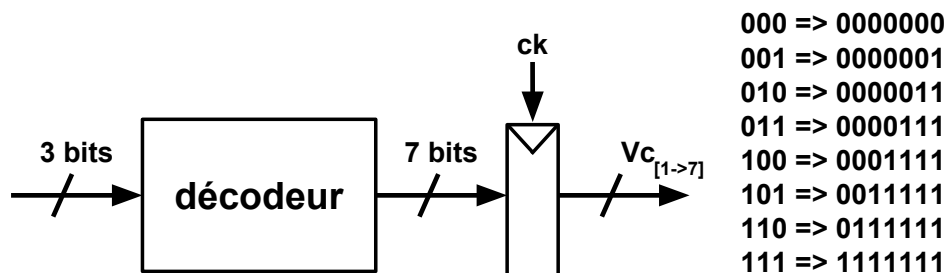


FIGURE 6.19 – Décodeur numérique contrôlant la tension d’entrée VC de la transconductance différentielle

Pour cet exemple, nous proposons le placement présenté sur la figure 6.20. Les zones en bleu sont symétriques, la zone *Hsymétrique* est le symétrique de la zone bleue située à gauche. Pour raisons de clarté, sa représentation dans le *slicing tree* est simplifiée sur la figure 6.21 représentant le *slicing tree* relatif à ce placement. La description du script *Python* de ce circuit est décrite par le code A.3 se trouvant en annexe. On peut y observer la description des modules comprenant leurs noms, leurs dimensions et la description des *nets* du circuit.

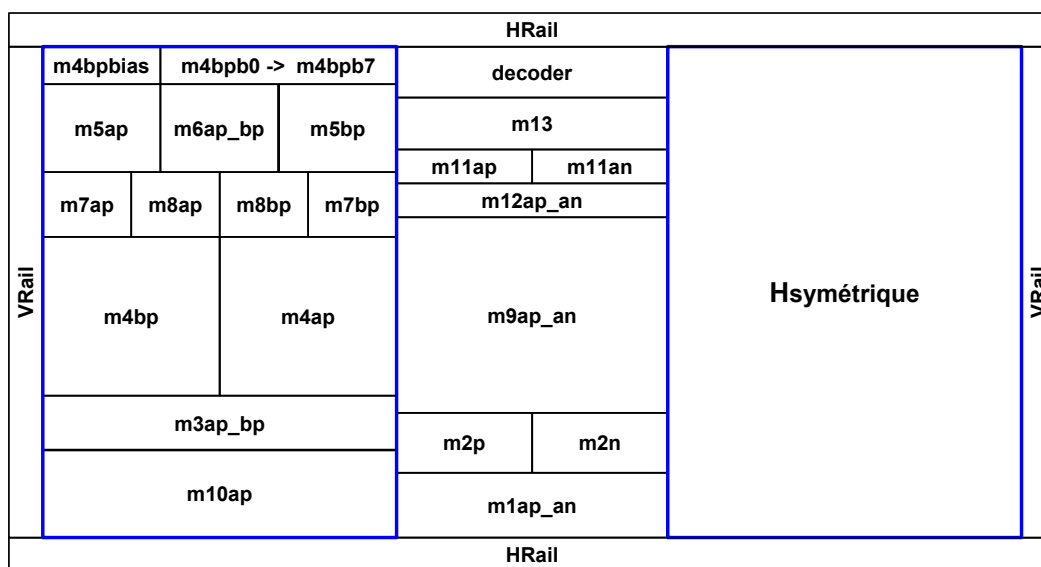


FIGURE 6.20 – Placement de la transconductance différentielle configurable

La description du *slicing tree* de la figure 6.21 est présentée par le code A.3. Le nombre de facteurs de forme pour chacun des modules est le suivant :

- Pour 32 modules : 2 facteurs de forme.
- Pour les 17 autres modules : un facteur de forme.

Suite à l’exécution du script *Python*, l’algorithme de placement calcule l’ensemble des facteurs de forme possibles. Une fois le nombre de facteurs de forme déterminé, les résultats sont présentés sous la forme d’un graphe de point illustré par la figure 6.22.

Quelques informations des résultats de placements possibles sont présentées ci-dessous :

- **Nombre totale de placements valides** : 263 placements.

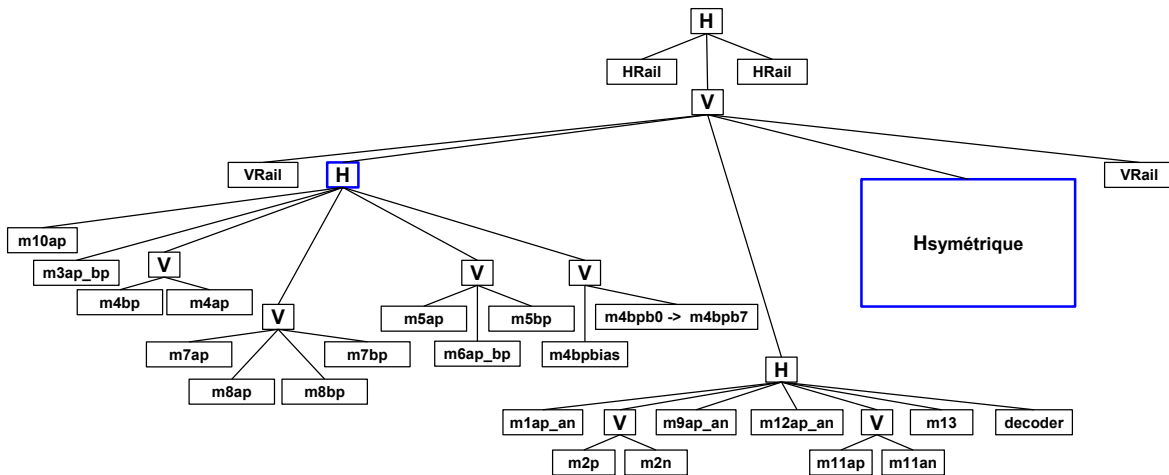


FIGURE 6.21 – *Slicing tree* du placement de la transconductance différentielle configurable

- **Placement occupant le moins de surface** : $9630,7 \mu m^2$.
- **Placement occupant le plus de surface** : $14990,1 \mu m^2$.
- **Temps d'exécution du placement** : 1,382 secondes.

En tant qu'exemple, on choisit le circuit de plus petite surface auquel on applique le routage global et le routage détaillé. Le résultat obtenu est présenté par la figure 6.23. On peut clairement y observer les deux parties symétriques selon l'axe verticale au niveau du centre du circuit.

Quelques informations à propos du routage global et du routage détaillé sont présentées ci-dessous :

- **Nombre de GCells** : 257 GCells.
- **Temps d'exécution du routage global** : 0,061 secondes.
- **Surface suite à l'étirement des canaux** : $15256,6 \mu m^2$.
- **Longueur de fil** : $41751 \mu m$.
- **Nombre de segments** : 891 segments.
- **Nombre de contacts** : 1025 contacts.
- **Temps d'exécution du routage détaillé** : 0,189 secondes.

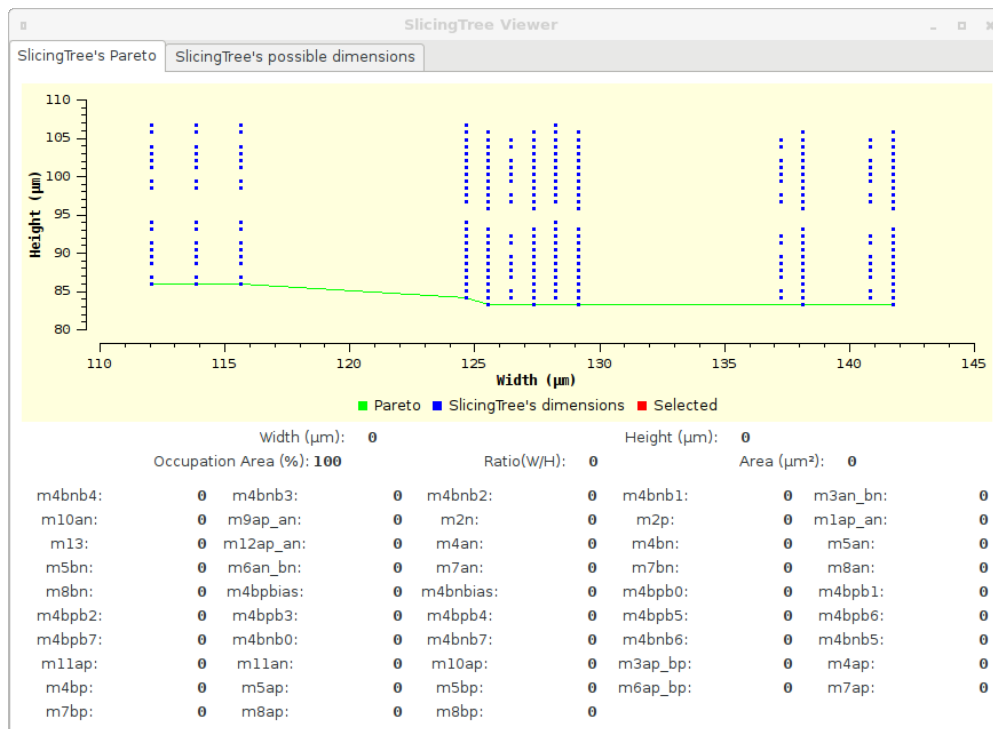


FIGURE 6.22 – Graphe de points des placements possibles

6.5 Conclusion

Notre approche de placement et de routage de circuits analogiques et mixtes a été expérimentée sur trois circuits de tailles différentes avec des contraintes de placement et de routage variées. Dans la réalisation du dessin des masques de ces trois circuits, les concepteurs ont la tâche de définir le placement relatif du circuit. La description du placement relatif est accompagnée de contraintes de placement et de routage spécifiées par le concepteur. À partir de ces informations, notre outil de placement propose au concepteur un ensemble de possibilités de placement conséquent (80 ou 90 différents pour le premier circuit, 19 pour le second et 263 pour le troisième). Chacun de ces placements peut être examiné par le concepteur et la génération de l'un d'entre eux est de l'ordre de quelques secondes (1,523 secondes pour le troisième circuit pour un placement) laissant la possibilité aux concepteurs d'essayer plusieurs solutions. Une fois le choix d'un placement défini, les phases de routage global et de routage détaillé sont exécutées en un temps court (jusqu'à 0,318 secondes pour le troisième circuit) permettant d'essayer plusieurs solutions rapidement. La description de ces circuits à partir de scripts *Python* permettent d'automatiser l'exécution de ces étapes et un ajustement rapide des modules du circuit ou des contraintes de la phase de placement et de routage.

6.6 Références

- [1] Alain Greiner and François Pêcheux. *Alliance : A complete set of CAD tools for teaching vlsi design*. 1992. 154
- [2] Jacky Porte. *OCEANE : Outils pour la Conception et l'Enseignement des circuits intégrés ANalogiques – reference manual*, January 2018. 154

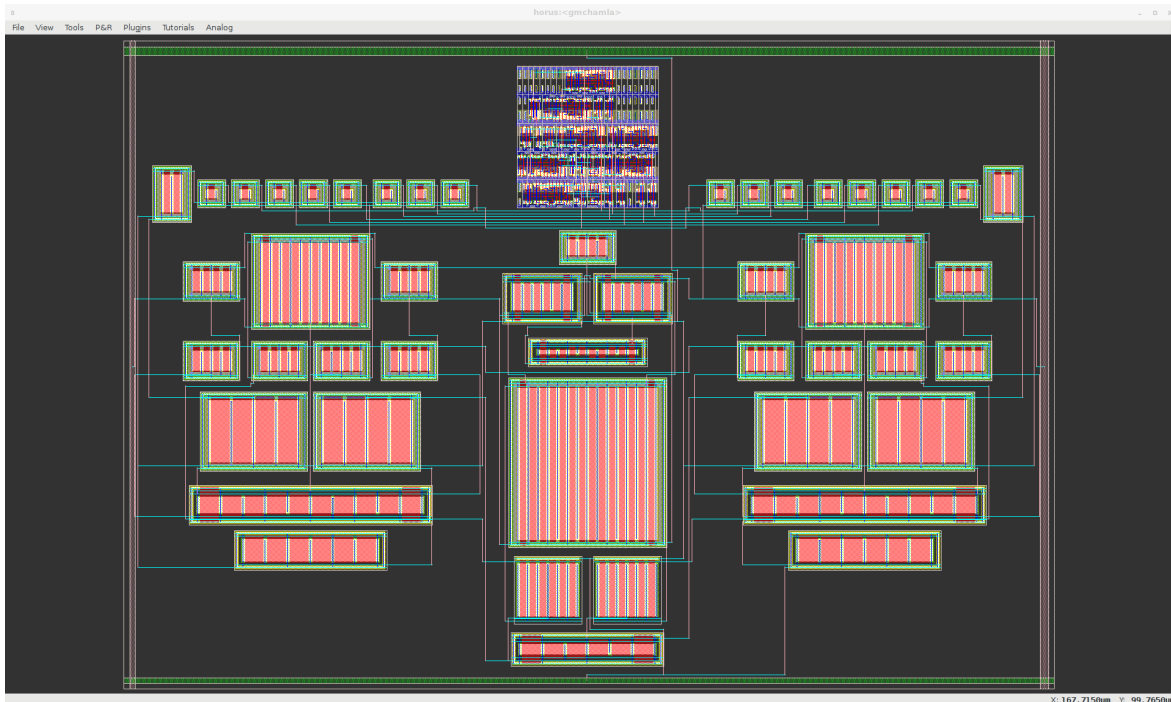


FIGURE 6.23 – Transconductance différentielle configurable placée et routée

- [3] Christian Masson Gabriel Gouvine Sophie Belloeil Damien Dupuis Christophe Alexandre Hugo Clement Marek Sroka Jean-Paul Chaput, Rémy Escassut and Wu Yifei. Coriolis, 2018. [155](#)
- [4] David Chamla, Andreas Kaiser, Andreia Cathelin, and Didier Belot. Ag/sub m/-c low-pass filter for zero-if mobile applications with a very wide tuning range. *IEEE Journal of Solid-State Circuits*, 40(7) :1443–1450, 2005. [166](#)

Chapitre 7

Conclusion et perspectives

Sommaire

7.1 Conclusions	172
7.1.1 Perspectives	173

7.1 Conclusions

Dans ces travaux de thèse, nous avons proposé une solution pour répondre à la problématique du dessin des masques lors de la conception de circuits analogiques et mixtes. La conception des circuits analogiques est le facteur limitant de la conception de circuits mixtes. Jusqu'à présent, les outils logiciels n'étaient pas adaptés à l'interprétation conjointe des contraintes du monde numérique et du monde analogique. Les deux objectifs principaux de cette thèse consistent à définir une méthodologie permettant d'automatiser le placement et le routage de la partie analogique et en unifiant le routage numérique et analogique.

État de l'art des outils de CAO pour circuits analogiques et mixtes - Chapitre 2

Ce chapitre présente l'ensemble des travaux précédemment réalisés et dédiés à l'automatisation des circuits analogiques et mixtes. L'automatisation du placement et du routage des circuits analogiques est un sujet qui a été traité par de nombreuses études au cours de ces deux dernières décennies. Ces études présentent différentes approches de conception, basées sur des outils procéduraux, des outils utilisant des méthodes d'optimisation ou encore se basant des *templates*. Cette présentation de l'état de l'art nous permet d'identifier les avantages et les limitations des études existantes. L'historique des travaux dédiés aux outils de CAO pour la conception de circuits analogiques présente les éléments déjà disponibles pour réaliser notre flot de conception mixte.

Placement - Chapitre 3

Ce chapitre décrit notre approche de placement analogique et mixte qui propose une méthodologie basée sur des interactions entre les concepteurs et l'outil de placement. L'automatisation du placement a pour objectif de placer l'ensemble des modules du circuit selon un placement relatif décrit par les concepteurs en utilisant une représentation en *slicing tree*. De cette manière, les concepteurs gardent un contrôle sur le résultat du placement. La description du *slicing tree* est décrite à partir d'un script *Python* dans lequel les concepteurs précisent également les facteurs de forme. À partir des contraintes fournies, l'outil de placement calcule l'ensemble des facteurs de forme pouvant être obtenu et les concepteurs ont la possibilité de visualiser et essayer ces solutions de placement à travers une interface graphique. La génération d'un placement est exécutée en quelques secondes permettant donc aux concepteurs de pouvoir rapidement essayer plusieurs placements et de choisir le placement qui leur semble le plus adéquat.

Routage global - Chapitre 4

La phase de routage global consiste à définir les chemins grossiers des fils de routage pour chacun des *nets* du circuit. Le résultat du placement est transformé en un graphe de routage représentant les ressources de routage définies par les modules et les espaces vides du circuit. On utilise un algorithme de Dijkstra modifié dans le but de rechercher les arbres d'interconnexions les plus courts, les contraintes et les méthodes d'estimation de longueurs de fils sont adaptées en fonction du type de *net* traité. Dans le cas d'un *net* numérique, l'estimation des distances est moins précise et la fenêtre d'exploration restreinte afin de limiter le volume de données à parcourir. Dans le cas d'un *net* analogique, l'estimation des distances est plus précise et nécessite la gestion d'obstacle. La phase de rou-

tage est considérée mixte car le traitement des *nets* numériques et des *nets* analogiques suit un flot de conception identique. Le résultat du routage global entraîne un redimensionnement des canaux de routage de la partie analogique du circuit de telle sorte à ce que les fils de routage puissent avoir assez d'espace pour pouvoir être placés en respectant les règles de dessin.

Routage détaillé - Chapitre 5

La phase de routage détaillé est la phase exploitant le résultat du routage global pour construire les fils de routage. Cette phase comporte deux phases qui consistent à construire les fils de routage et ensuite à résoudre les problèmes de superposition. La construction des fils de routage consiste également à choisir les éléments de base pouvant être manipulés par le routeur détaillé. L'étape de résolution des superpositions de fils est fondée une approche *rip-up and reroute* caractérisée par une machine à états contrôlant le placement des fils de routage, l'invalidation et le remplacement d'un fil et la création de *dogleg*. Ce routeur détaillé extrêmement configurable traite simultanément les fils de routage numériques et analogiques tout en préservant les contraintes de routage spécifiées telles que les contraintes de symétrie.

Résultats - Chapitre 6

Les résultats obtenus en appliquant notre méthodologie sur trois circuits analogiques ou mixtes démontrent la capacité de notre outil à générer le dessin des masques d'un circuit complet en un temps court. Pour chacun de ces circuits, une fois le script *Python* décrit, le calcul des facteurs de forme de la phase de placement et l'exécution des deux phases de routage sont de l'ordre de plusieurs secondes ce qui est bien plus rapide qu'une approche de conception manuelle. De plus, les concepteurs ont la possibilité de router plusieurs solutions de placement en un temps tout aussi court ce qui s'avère un avantage conséquent. L'automatisation des étapes de placement et de routage s'insère dans une boucle d'itération entre le dimensionnement et la génération du dessin des masques, la capacité à générer rapidement le dessin des masques d'un circuit mixte est nécessaire pour pouvoir effectuer des ajustements en un temps court.

7.1.1 Perspectives

Dans ce manuscrit de thèse, nous avons expérimenté un flot de conception pour circuits mixtes en ayant objectif d'unifier le flot de conception numérique et analogique. À partir de l'expérience en conception de circuits et des précédents outils logiciels développés au LIP6 au cours de ces vingt dernières années, une méthodologie de placement routage mixte a été proposée. À notre connaissance, aucun routeur global et routeur détaillé ne sont capables de gérer les contraintes du monde numérique et du monde analogique telles que nous les prenons en compte. Les résultats obtenus dans le cadre de cette thèse sont encourageants et permettent de se rapprocher davantage d'une conception de circuits analogiques et mixtes automatisée. Dans la continuité de ces travaux, il serait intéressant d'introduire ou d'améliorer différents points :

Ajout des capacités, des inductances et des résistances

Dans l'état actuel de l'extension de Coriolis, les seuls modules considérés sont des modules comprenant un ou plusieurs transistors réalisant une fonction de base analogique. Or il est nécessaire de pouvoir incorporer les capacités, les inductances et les résistances afin de permettre la description d'une plus grande variété de circuits. Par exemple, des circuits tels que les circuits fonctionnant à haute fréquence (circuits radiofréquence) nécessitent l'intégration d'inductance. La génération du dessin des masques configurables de ces composants nécessite d'établir une méthodologie de construction générique.

Problème de caractérisation électrique des dessin des masques précédemment développés

La génération du dessin des masques des modules analogiques a été réalisée en se fondant sur des travaux de thèse précédents [?] et qui ont été améliorés. En effet, les dimensions, la position des fils de routage internes et la position des connecteurs des modules ont été corrigées de telle sorte à ce qu'elles soient positionnées sur la grille du routeur détaillé. Cette grille correspond aux positions que les fils de routage peuvent prendre lorsqu'ils sont placés par le routeur détaillé et ces positions garantissent le respect des règles dessins. Cet ajustement nécessite d'être appliqué à tous les styles de dessin des masques précédemment définis.

Enrichissement de l'interface graphique avec un éditeur de *slicing tree*

Lors de la phase de placement, les concepteurs ont la tâche de définir le placement relatif des modules sous la forme d'un *slicing tree*. Cette description doit être décrite au sein d'un script *python* suivant la syntaxe détaillée dans le chapitre 3. Néanmoins, cette représentation en *slicing tree* peut s'avérer complexe pour un concepteur qui ne serait pas familier avec cette représentation. C'est pourquoi, un éditeur de *slicing tree* permettrait aux concepteurs de placer manuellement les modules de son circuit et l'éditeur présenterait en parallèle la correspondance du placement induit sous la forme d'un *slicing tree*. On pourrait également imaginer que les concepteurs pourraient construire un *slicing tree* et l'éditeur en présenterait la correspondance en termes de placement de manière interactive.

Corrections des erreurs de DRC

Avec les modifications apportées aux modules analogiques, des erreurs de règles de dessins ont été introduites provoquant des erreurs lors de l'étape vérification (Design Rules Check). Ces erreurs interviennent dans le positionnement des segments de métaux 1 et 2 et dépendent également des paramètres de génération du module analogique. Des erreurs supplémentaires apparaissent dans la phase du routage détaillé, il arrive que la position des contacts soient trop proches. Ces erreurs ont été analysées et localisées au niveau des classes concernées, le temps imparti pour cette thèse ne permettait pas de corriger ces erreurs.

Extensions aux applications liées à la photonique

Dans la continuité de l'ajout des capacités, des inductances et des résistances, un projet actuel vise à étendre les outils de Coriolis à la génération du dessin des masques pour des applications dédiées à la photonique. La construction des transistors et des fils de

routage d'un circuit CMOS consiste à assembler des régions rectangulaires. En photonique, il est commun d'avoir des structures plus complexes devant être définies par un polygone. L'intégration de ces polygones est actuellement en cours dans le cadre d'une collaboration avec des chercheurs de l'université ETH Zurich.

Rassemblement d'une communauté autour de nos logiciels libres

Un des aspects majeurs de la chaîne d'outils logiciels développée au LIP6 repose sur le concept du logiciel libre introduit par Richard Stallman. Le concept du logiciel libre consiste à permettre aux utilisateurs d'avoir le contrôle complet du logiciel qu'ils utilisent. Autrement dit, les utilisateurs sont libres d'utiliser ce logiciel, d'avoir accès à l'ensemble du code source du logiciel et de le modifier en cas de nécessité. Cela implique une collaboration entre utilisateurs et développeurs formant une communauté autour du logiciel libre.

Au-delà du contrôle du logiciel libre par ses utilisateurs, l'objectif consiste à permettre à des concepteurs de ne pas être soumis à des accords de non-divulgateion. Actuellement, si le dessin des masques d'un circuit a été conçu à partir d'outils logiciels industriels, il ne peut être légalement partagé publiquement. Par conséquent, le partage des circuits et le partage des connaissances en sont limités. Avec notre chaîne de placement routage destinée aux circuits numériques, aux circuits analogiques et aux circuits mixtes, nous souhaitons permettre aux concepteurs de publier leurs circuits conçus avec nos logiciels libres et en utilisant une technologie libre. Ce critère de liberté et de partage des connaissances du domaine de la conception de circuit font partie des lignes directrices de la recherche au sein du laboratoire d'informatique de Paris 6.

Chapitre 8

Publications

8.1 Références

- [1] Eric Lao, Marie-Minerve Louërat and Jean-Paul Chaput Highly configurable place and route for analog and mixed-signal circuits In *PhD Forum at Design, Automation and Test in Europe Conference (DATE), Lausanne, Switzerland (2017)*.
- [2] Eric Lao, Marie-Minerve Louërat and Jean-Paul Chaput Semi-automated analog placement In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS), Monte Carlo, Monaco, 2016*, pp. 432-433.
- [3] Eric Lao, Marie-Minerve Louërat and Jean-Paul Chaput Semi-Automated Analog Placement based on Margin Tolerances In *the 20th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI 2016), Kyoto, Japan (2016)*.

Annexe A

Annexes

A.1 Script de l'amplificateur à source de courant ajustable

Code A.1 – Script *Python* de l'amplificateur à source de courant ajustable - Description des modules et des *nets*

```
1 self.devicesSpecs = \  
2  
3 # | Class          | Instance | Layout Style | Type | W      | L      | Dnm | SFirst | Bulk | BulkC |  
4 # +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
5 [ [ Transistor , 'mn1OTA', 'WIP Transistor', NMOS, 51.95 , 6.825 , 0, True , 0xf, False ]  
6 , [ Transistor , 'mn2OTA', 'WIP Transistor', NMOS, 51.9 , 6.825 , 0, True , 0xf, False ]  
7 , [ Transistor , 'mp3OTA', 'WIP Transistor', PMOS, 9.0 , 6.825 , 0, True , 0xf, True ]  
8 , [ Transistor , 'mp4OTA', 'WIP Transistor', PMOS, 9.0 , 6.825 , 0, True , 0xf, True ]  
9 , [ Transistor , 'mn5OTA', 'WIP Transistor', NMOS, 110.6 , 6.825 , 0, True , 0xf, True ]  
10  
11 , [ Transistor , 'mnp11', 'WIP Transistor', NMOS, 55.3 , 6.825 , 0, True , 0xf, True ]  
12 , [ Transistor , 'mn1', 'WIP Transistor', NMOS, 1268.9 , 3.50 , 0, True , 0xf, False ]  
13 , [ Transistor , 'mn2', 'WIP Transistor', NMOS, 317.225, 3.5 , 0, True , 0xf, True ]  
14 , [ Transistor , 'mn1r', 'WIP Transistor', NMOS, 1.53 , 3.5 , 0, True , 0xf, True ]  
15 , [ Transistor , 'mn1r', 'WIP Transistor', NMOS, 3.6 , 3.5 , 0, True , 0xf, True ]  
16 , [ Transistor , 'mp3', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
17 , [ Transistor , 'mp4', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
18  
19 , [ Transistor , 'mp3c', 'WIP Transistor', PMOS, 0.5 , 15.2 , 0, True , 0xf, False ]  
20 , [ Transistor , 'mp4c', 'WIP Transistor', PMOS, 0.5 , 10.625, 0, True , 0xf, False ]  
21 , [ Transistor , 'mp14', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
22 , [ Transistor , 'mp14c', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]  
23  
24 , [ Transistor , 'mp21', 'WIP Transistor', PMOS, 2.4 , 3.5 , 0, True , 0xf, True ]  
25 , [ Transistor , 'mp22', 'WIP Transistor', PMOS, 2.4 , 3.5 , 0, True , 0xf, True ]  
26 , [ Transistor , 'mp23', 'WIP Transistor', PMOS, 2.4 , 3.5 , 0, True , 0xf, True ]  
27 , [ Transistor , 'mp24', 'WIP Transistor', PMOS, 2.4 , 3.5 , 0, True , 0xf, True ]  
28  
29 , [ Transistor , 'mpc21', 'WIP Transistor', PMOS, 2.0 , 15.175, 0, True , 0xf, False ]  
30 , [ Transistor , 'mpc22', 'WIP Transistor', PMOS, 2.0 , 15.175, 0, True , 0xf, False ]  
31 , [ Transistor , 'mpc23', 'WIP Transistor', PMOS, 2.0 , 15.175, 0, True , 0xf, False ]  
32 , [ Transistor , 'mpc24', 'WIP Transistor', PMOS, 2.0 , 15.175, 0, True , 0xf, False ]  
33  
34 , [ Transistor , 'mp251', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
35 , [ Transistor , 'mp252', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
36 , [ Transistor , 'mp253', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
37 , [ Transistor , 'mp254', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
38 , [ Transistor , 'mp261', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
39 , [ Transistor , 'mp262', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
40 , [ Transistor , 'mp263', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
41 , [ Transistor , 'mp264', 'WIP Transistor', PMOS, 0.6 , 3.5 , 0, True , 0xf, True ]  
42  
43 , [ Transistor , 'mpc251', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]  
44 , [ Transistor , 'mpc252', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]  
45 , [ Transistor , 'mpc253', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]  
46 , [ Transistor , 'mpc254', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]  
47 , [ Transistor , 'mpc261', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]  
48 , [ Transistor , 'mpc262', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]
```

```

49 , [ Transistor , 'mpc263', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]
50 , [ Transistor , 'mpc264', 'WIP Transistor', PMOS, 0.5 , 15.175, 0, True , 0xf, False ]
51 , [ decoder , 'decoder' ]
52 ]
53 self.netTypes = \
54 # | Net | Type |
55 # +-----+-----+
56 { 'ep' : { 'isExternal' : True }
57 , 'em' : { 'isExternal' : True }
58 , 'va' : { 'isExternal' : False }
59 , 'sp' : { 'isExternal' : True }
60 , 'evp1' : { 'isExternal' : False }
61 , 'vp' : { 'isExternal' : True }
62 , 'd14' : { 'isExternal' : False }
63 , 'd4' : { 'isExternal' : False }
64 , 'd3' : { 'isExternal' : False }
65 , 'd21' : { 'isExternal' : False }
66 , 'd22' : { 'isExternal' : False }
67 , 'd23' : { 'isExternal' : False }
68 , 'd24' : { 'isExternal' : False }
69 , 'd251' : { 'isExternal' : False }
70 , 'd252' : { 'isExternal' : False }
71 , 'd253' : { 'isExternal' : False }
72 , 'd254' : { 'isExternal' : False }
73 , 'd261' : { 'isExternal' : False }
74 , 'd262' : { 'isExternal' : False }
75 , 'd263' : { 'isExternal' : False }
76 , 'd264' : { 'isExternal' : False }
77 , 'dg11' : { 'isExternal' : False }
78 , 'dg2' : { 'isExternal' : False }
79 , 'd1r' : { 'isExternal' : False }
80 , 'vdd' : { 'isExternal' : True }
81 , 'vss' : { 'isExternal' : True }
82
83 # gate voltage of transistors for digital control of biasing VC
84 , 'vz' : { 'isExternal' : False }
85 , 'vc0' : { 'isExternal' : False }
86 , 'vc1' : { 'isExternal' : False }
87 , 'vc2' : { 'isExternal' : False }
88 , 'vc3' : { 'isExternal' : False }
89 , 'vc4' : { 'isExternal' : False }
90 , 'vc5' : { 'isExternal' : False }
91 , 'vc6' : { 'isExternal' : False }
92 , 'vc7' : { 'isExternal' : False }
93 # gate voltage of transistors for digital control of biasing VC
94 , 'cmd0' : { 'isExternal':True}
95 , 'cmd1' : { 'isExternal':True}
96 , 'cmd2' : { 'isExternal':True}
97 , 'cmd3' : { 'isExternal':True}
98 }
99
100 self.netSpecs = \
101 # | Net | Connector |
102 # +-----+-----+
103 { 'ep' : [ ('mn1O1A', 'G') ]
104 , 'em' : [ ('mn2O1A', 'G') ]
105 , 'va' : [ ('mn1O1A', 'D'), ('mp3O1A', 'D'), ('mp3O1A', 'G'), ('mp4O1A', 'G') ]
106 , 'sp' : [ ('mn2O1A', 'D'), ('mp4O1A', 'D') ]
107 , 'vz' : [ ('mn5O1A', 'D'), ('mn1O1A', 'S'), ('mn2O1A', 'S') ]
108 , 'vc0' : [ ('mpc251', 'G'), ('decoder', 'vc(0)') ]
109 , 'vc1' : [ ('mpc252', 'G'), ('decoder', 'vc(1)') ]
110 , 'vc2' : [ ('mpc253', 'G'), ('decoder', 'vc(2)') ]
111 , 'vc3' : [ ('mpc254', 'G'), ('decoder', 'vc(3)') ]
112 , 'vc4' : [ ('mpc261', 'G'), ('decoder', 'vc(4)') ]
113 , 'vc5' : [ ('mpc262', 'G'), ('decoder', 'vc(5)') ]
114 , 'vc6' : [ ('mpc263', 'G'), ('decoder', 'vc(6)') ]
115 , 'vc7' : [ ('mpc264', 'G'), ('decoder', 'vc(7)') ]
116 , 'evp1' : [ ('mn5O1A', 'G'), ('mnp11', 'G'), ('mnp11', 'D')
117 , ('mpc21', 'D'), ('mpc22', 'D'), ('mpc23', 'D'), ('mpc24', 'D')
118 , ('mpc251', 'D'), ('mpc252', 'D'), ('mpc253', 'D'), ('mpc254', 'D')
119 , ('mpc261', 'D'), ('mpc262', 'D'), ('mpc263', 'D'), ('mpc264', 'D') ]
120 , 'vp' : [ ('mp14', 'G'), ('mp4', 'G'), ('mp3', 'G'), ('mp4c', 'D'), ('mnl', 'D')
121 , ('mp21', 'G'), ('mp22', 'G'), ('mp23', 'G'), ('mp24', 'G')
122 , ('mp251', 'G'), ('mp252', 'G'), ('mp253', 'G'), ('mp254', 'G')
123 , ('mp261', 'G'), ('mp262', 'G'), ('mp263', 'G'), ('mp264', 'G') ]

```

A.1. SCRIPT DE L'AMPLIFICATEUR À SOURCE DE COURANT AJUSTABLE

```

124 , 'd14' : [ ('mp14' , 'D'), ('mp14c' , 'S') ]
125 , 'd4' : [ ('mp4' , 'D'), ('mp4c' , 'S') ]
126 , 'd3' : [ ('mp3' , 'D'), ('mp3c' , 'S') ]
127 , 'd21' : [ ('mp21' , 'D'), ('mpc21' , 'S') ]
128 , 'd22' : [ ('mp22' , 'D'), ('mpc22' , 'S') ]
129 , 'd23' : [ ('mp23' , 'D'), ('mpc23' , 'S') ]
130 , 'd24' : [ ('mp24' , 'D'), ('mpc24' , 'S') ]
131 , 'd251' : [ ('mp251' , 'D'), ('mpc251' , 'S') ]
132 , 'd252' : [ ('mp252' , 'D'), ('mpc252' , 'S') ]
133 , 'd253' : [ ('mp253' , 'D'), ('mpc253' , 'S') ]
134 , 'd254' : [ ('mp254' , 'D'), ('mpc254' , 'S') ]
135 , 'd261' : [ ('mp261' , 'D'), ('mpc261' , 'S') ]
136 , 'd262' : [ ('mp262' , 'D'), ('mpc262' , 'S') ]
137 , 'd263' : [ ('mp263' , 'D'), ('mpc263' , 'S') ]
138 , 'd264' : [ ('mp264' , 'D'), ('mpc264' , 'S') ]
139 , 'dg11' : [ ('mn11' , 'D'), ('mn11' , 'G'), ('mn1r' , 'G'), ('mp14c' , 'D') ]
140 , 'dg2' : [ ('mn1' , 'G'), ('mn2' , 'G'), ('mn2' , 'D') ]
141 , 'dir' : [ ('mn1' , 'S'), ('mn1r' , 'D') ]
142 , 'vdd' : [ ('mp3OTA' , 'S'), ('mp4OTA' , 'S')
143 , ('mp14' , 'S'), ('mp14c' , 'B'), ('mp4' , 'S'), ('mp4c' , 'B')
144 , ('mp3' , 'S'), ('mp3c' , 'B'), ('mp21' , 'S'), ('mp22' , 'S')
145 , ('mp23' , 'S'), ('mp24' , 'S'), ('mp251' , 'S'), ('mp252' , 'S')
146 , ('mp253' , 'S'), ('mp254' , 'S'), ('mp261' , 'S'), ('mp262' , 'S')
147 , ('mp263' , 'S'), ('mp264' , 'S'), ('mpc21' , 'B'), ('mpc22' , 'B')
148 , ('mpc23' , 'B'), ('mpc24' , 'B'), ('mpc251' , 'B'), ('mpc252' , 'B')
149 , ('mpc253' , 'B'), ('mpc254' , 'B'), ('mpc261' , 'B'), ('mpc262' , 'B')
150 , ('mpc263' , 'B'), ('mpc264' , 'B') ]
151 , 'vss' : [ ('mn1OTA' , 'B'), ('mn2OTA' , 'B'), ('mn5OTA' , 'S'), ('mn1' , 'B')
152 , ('mp4c' , 'G'), ('mp3c' , 'G'), ('mpc21' , 'G'), ('mpc22' , 'G')
153 , ('mpc23' , 'G'), ('mpc24' , 'G'), ('mn11' , 'S'), ('mn1r' , 'S'), ('mn2' , 'S') ]
154 , 'cmd0' : [ ('decoder' , 'command(0)' ) ]
155 , 'cmd1' : [ ('decoder' , 'command(1)' ) ]
156 , 'cmd2' : [ ('decoder' , 'command(2)' ) ]
157 , 'cmd3' : [ ('decoder' , 'command(3)' ) ]
158 }

```

Code A.2 – Script *Python* de l'amplificateur à source de courant ajustable - Description du *slicing tree*

```

1
2 self.readParameters( './../oceane/all_ota_t.txt' )
3
4 self.beginCell( 'ota' )
5 self.doDevices()
6 self.doNets ()
7
8 self.beginSlicingTree()
9
10 self.pushHNode( Center )
11 self.addHRail( self.getNet( 'vss' ), 'METAL4', 2, "CHI", "IH1" )
12
13 self.pushVNode( Center )
14 self.addDevice( 'decoder' , Center )
15
16 self.pushHNode( Center )
17 self.addDevice( 'mn1' , Center, span=(24.0, 4.0, 12.0))
18
19 self.pushVNode( Center )
20 self.pushHNode( Center )
21 self.addDevice( 'mn11' , Center, span=(2.0, 2.0, 2.0) )
22 self.addDevice( 'mn1r' , Center, span=(2.0, 2.0, 2.0) )
23 self.popNode()
24
25 self.addDevice( 'mn2' , Center, span=(16.0, 2.0, 8.0) )
26 self.addDevice( 'mnp11' , Center, span=( 2.0, 2.0, 2.0) )
27 self.popNode()
28
29 self.pushVNode( Center )
30 self.addDevice( 'mpc251' , Center, span=(1.0, 1.0, 1.0) )
31 self.addDevice( 'mpc252' , Center, span=(1.0, 1.0, 1.0) )
32 self.addDevice( 'mpc253' , Center, span=(1.0, 1.0, 1.0) )
33 self.addDevice( 'mpc254' , Center, span=(1.0, 1.0, 1.0) )
34 self.addDevice( 'mpc261' , Center, span=(1.0, 1.0, 1.0) )
35 self.addDevice( 'mpc262' , Center, span=(1.0, 1.0, 1.0) )

```

```

36 self.addDevice( 'mpc263', Center, span=(1.0, 1.0, 1.0) )
37 self.addDevice( 'mpc264', Center, span=(1.0, 1.0, 1.0) )
38 self.popNode()
39
40 self.pushVNode( Center )
41 self.addDevice( 'mp251', Center, span=(1.0, 1.0, 1.0) )
42 self.addDevice( 'mp252', Center, span=(1.0, 1.0, 1.0) )
43 self.addDevice( 'mp253', Center, span=(1.0, 1.0, 1.0) )
44 self.addDevice( 'mp254', Center, span=(1.0, 1.0, 1.0) )
45 self.addDevice( 'mp261', Center, span=(1.0, 1.0, 1.0) )
46 self.addDevice( 'mp262', Center, span=(1.0, 1.0, 1.0) )
47 self.addDevice( 'mp263', Center, span=(1.0, 1.0, 1.0) )
48 self.addDevice( 'mp264', Center, span=(1.0, 1.0, 1.0) )
49 self.popNode()
50
51 self.pushVNode( Center )
52 self.addDevice( 'mp14c', Center, span=(1.0, 1.0, 1.0) )
53 self.addDevice( 'mp4c', Center, span=(1.0, 1.0, 1.0) )
54 self.addDevice( 'mp3c', Center, span=(1.0, 1.0, 1.0) )
55 self.addDevice( 'mpc21', Center, span=(4.0, 1.0, 1.0) )
56 self.popNode()
57
58 self.pushVNode( Center )
59 self.addDevice( 'mpc22', Center, span=(4.0, 1.0, 1.0) )
60 self.addDevice( 'mpc23', Center, span=(4.0, 1.0, 1.0) )
61 self.addDevice( 'mpc24', Center, span=(4.0, 1.0, 1.0) )
62 self.popNode()
63
64 self.pushVNode( Center )
65 self.addDevice( 'mp14', Center, span=(1.0, 1.0, 1.0) )
66 self.addDevice( 'mp4', Center, span=(1.0, 1.0, 1.0) )
67 self.addDevice( 'mp3', Center, span=(1.0, 1.0, 1.0) )
68 self.addDevice( 'mp21', Center, span=(4.0, 1.0, 1.0) )
69 self.addDevice( 'mp22', Center, span=(4.0, 1.0, 1.0) )
70 self.addDevice( 'mp23', Center, span=(4.0, 1.0, 1.0) )
71 self.addDevice( 'mp24', Center, span=(4.0, 1.0, 1.0) )
72 self.popNode()
73
74 self.popNode()
75
76 # OTA
77 self.pushHNode( Center )
78 self.addDevice( 'mn5OTA', Center, span=(4.0, 1.0, 1.0) )
79
80 self.pushVNode( Center )
81 self.addSymmetry( 0, 1 )
82 self.addDevice( 'mn1OTA', Center, span=(2.0, 1.0, 1.0) )
83 self.addDevice( 'mn2OTA', Center, span=(2.0, 1.0, 1.0) )
84 self.popNode()
85
86 self.pushVNode( Center )
87 self.addSymmetry( 0, 1 )
88 self.addDevice( 'mp3OTA', Center, span=(2.0, 1.0, 1.0) )
89 self.addDevice( 'mp4OTA', Center, span=(2.0, 1.0, 1.0) )
90 self.popNode()
91
92 self.popNode()
93 self.popNode()
94
95 self.addHRail( self.getNet('vdd'), 'METAL4', 2, "CH2", "IH2" )
96
97 self.popNode()
98 self.endSlicingTree()
99 self.endCell()

```

A.2 Script de la transconductance différentielle contrôlable

Code A.3 – Script *Python* de la transconductance différentielle contrôlable - Description des modules et des *nets*

A.2. SCRIPT DE LA TRANSCONDUCTANCE DIFFÉRENTIELLE CONTRÔLABLE

```

1 self.devicesSpecs = \
2 # | Class | Instance | Layout Style | Type | W | L | Dnm | SFirst | Bulk | BulkC |
3 # +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4 [ [ Transistor , 'm10ap' , 'WIP Transistor' , NMOS, 21.44 , 3.00 , 0, True , 0xf, True ]
5 , [ DifferentialPair , 'm3ap_bp' , 'WIP DP' , NMOS, 10.56 , 3.00 , 1, True , 0xf, True ]
6 , [ Transistor , 'm4ap' , 'WIP Transistor' , PMOS, 39.76 , 3.00 , 0, True , 0xf, True ]
7 , [ Transistor , 'm4bp' , 'WIP Transistor' , PMOS, 39.76 , 3.00 , 0, True , 0xf, True ]
8 , [ Transistor , 'm5ap' , 'WIP Transistor' , PMOS, 12.92 , 1.00 , 0, True , 0xf, True ]
9 , [ Transistor , 'm5bp' , 'WIP Transistor' , PMOS, 12.92 , 1.00 , 0, True , 0xf, True ]
10 , [ CommonSourcePair , 'm6ap_bp' , 'WIP CSP' , PMOS, 50.40 , 1.00 , 1, True , 0xf, True ]
11 , [ Transistor , 'm7ap' , 'WIP Transistor' , PMOS, 12.92 , 1.00 , 0, True , 0xf, True ]
12 , [ Transistor , 'm7bp' , 'WIP Transistor' , PMOS, 12.92 , 1.00 , 0, True , 0xf, True ]
13 , [ Transistor , 'm8ap' , 'WIP Transistor' , PMOS, 12.73 , 1.00 , 0, True , 0xf, True ]
14 , [ Transistor , 'm8bp' , 'WIP Transistor' , PMOS, 12.73 , 1.00 , 0, True , 0xf, True ]
15 , [ DifferentialPair , 'm11an' , 'WIP DP' , NMOS, 8.57328, 1.00 , 1, True , 0xf, False ]
16 , [ DifferentialPair , 'm11ap' , 'WIP DP' , NMOS, 8.57328, 1.00 , 1, True , 0xf, False ]
17 , [ DifferentialPair , 'm12ap_an' , 'WIP DP' , NMOS, 2.94359, 1.00 , 1, True , 0xf, True ]
18 , [ Transistor , 'm13' , 'WIP Transistor' , PMOS, 9.95588, 1.00 , 0, True , 0xf, True ]
19 , [ DifferentialPair , 'm1ap_an' , 'WIP DP' , NMOS, 3.445 , 3.00 , 1, True , 0xf, True ]
20 , [ Transistor , 'm2p' , 'WIP Transistor' , NMOS, 50.44 , 1.00 , 0, True , 0x1, False ]
21 , [ Transistor , 'm2n' , 'WIP Transistor' , NMOS, 50.44 , 1.00 , 0, True , 0x1, False ]
22 , [ CommonSourcePair , 'm9ap_an' , 'WIP CSP' , PMOS, 145.46 , 1.00 , 1, True , 0xf, True ]
23 , [ Transistor , 'm10an' , 'WIP Transistor' , NMOS, 21.44 , 3.00 , 0, True , 0xf, True ]
24 , [ DifferentialPair , 'm3an_bn' , 'WIP DP' , NMOS, 10.56 , 3.00 , 1, True , 0xf, True ]
25 , [ Transistor , 'm4an' , 'WIP Transistor' , PMOS, 39.76 , 3.00 , 0, True , 0xf, True ]
26 , [ Transistor , 'm4bn' , 'WIP Transistor' , PMOS, 39.76 , 3.00 , 0, True , 0xf, True ]
27 , [ Transistor , 'm5an' , 'WIP Transistor' , PMOS, 12.92 , 1.00 , 0, True , 0xf, True ]
28 , [ Transistor , 'm5bn' , 'WIP Transistor' , PMOS, 12.92 , 1.00 , 0, True , 0xf, True ]
29 , [ CommonSourcePair , 'm6an_bn' , 'WIP CSP' , PMOS, 50.40 , 1.00 , 1, True , 0xf, True ]
30 , [ Transistor , 'm7an' , 'WIP Transistor' , PMOS, 12.92 , 1.00 , 0, True , 0xf, True ]
31 , [ Transistor , 'm7bn' , 'WIP Transistor' , PMOS, 12.92 , 1.00 , 0, True , 0xf, True ]
32 , [ Transistor , 'm8an' , 'WIP Transistor' , PMOS, 12.73 , 1.00 , 0, True , 0xf, True ]
33 , [ Transistor , 'm8bn' , 'WIP Transistor' , PMOS, 12.73 , 1.00 , 0, True , 0xf, True ]
34 # transistors for digital control of biasing VC
35 , [ Transistor , 'm4bpbias' , 'WIP Transistor' , PMOS, 12.00 , 1.00 , 0, True , 0xf, True ]
36 , [ Transistor , 'm4bnbias' , 'WIP Transistor' , PMOS, 12.00 , 1.00 , 0, True , 0xf, True ]
37 , [ Transistor , 'm4bpb0' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
38 , [ Transistor , 'm4bpb1' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
39 , [ Transistor , 'm4bpb2' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
40 , [ Transistor , 'm4bpb3' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
41 , [ Transistor , 'm4bpb4' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
42 , [ Transistor , 'm4bpb5' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
43 , [ Transistor , 'm4bpb6' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
44 , [ Transistor , 'm4bpb7' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
45 , [ Transistor , 'm4bnb0' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
46 , [ Transistor , 'm4bnb1' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
47 , [ Transistor , 'm4bnb2' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
48 , [ Transistor , 'm4bnb3' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
49 , [ Transistor , 'm4bnb4' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
50 , [ Transistor , 'm4bnb5' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
51 , [ Transistor , 'm4bnb6' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
52 , [ Transistor , 'm4bnb7' , 'WIP Transistor' , PMOS, 1.50 , 1.00 , 0, True , 0xf, True ]
53 , [ decoderModel , 'decoder' ]
54 ]
55
56 self.netTypes = \
57 # | Net | Type |
58 # +-----+-----+
59 { 'vin+' : { 'isExternal' : True }
60 , 'vin-' : { 'isExternal' : True }
61 , 'vout+' : { 'isExternal' : True }
62 , 'vout-' : { 'isExternal' : True }
63 , 'vout4+' : { 'isExternal' : True }
64 , 'vout4-' : { 'isExternal' : True }
65 , 'vref' : { 'isExternal' : True }
66 # vc becomes internal and has 2 signals vcp and vcn
67 , 'vcn' : { 'isExternal' : False }
68 , 'vcp' : { 'isExternal' : False }
69 , 'vb5' : { 'isExternal' : True }
70 , 'vb7' : { 'isExternal' : True }
71 , 'vb10an' : { 'isExternal' : True }
72 , 'vb10ap' : { 'isExternal' : True }
73 , 'vb12an' : { 'isExternal' : True }
74 , 'vb12ap' : { 'isExternal' : True }
75 , 'vb13' : { 'isExternal' : True }

```



```

76 , 'vdd' : { 'isExternal' : True }
77 , 'vss' : { 'isExternal' : True }
78 # gate voltage of transistors for digital control of biasing VC
79 , 'vc1' : { 'isExternal' : False }
80 , 'vc2' : { 'isExternal' : False }
81 , 'vc3' : { 'isExternal' : False }
82 , 'vc4' : { 'isExternal' : False }
83 , 'vc5' : { 'isExternal' : False }
84 , 'vc6' : { 'isExternal' : False }
85 , 'vc7' : { 'isExternal' : False }
86 }
87
88 self.netSpecs = \
89 # | Net | Connector |
90 # +-----+-----+
91 { 'vin+' : [ ('m1ap_an', 'G1') ]
92 , 'vin-' : [ ('m1ap_an', 'G2'), ]
93 , 'vout+' : [ ('m2n', 'D'), ('m9ap_an', 'D2'), ('m1lap', 'G1') ]
94 , 'vout-' : [ ('m2p', 'D'), ('m9ap_an', 'D1'), ('m1lan', 'G2') ]
95 # CMC.
96 , 'cmc_1' : [ ('m1lap', 'S'), ('m12ap_an', 'D1') ]
97 , 'cmc_1' : [ ('m1lan', 'S'), ('m12ap_an', 'D2') ]
98 # AmpLin N.
99 , 'ampn_1' : [ ('m6an_bn', 'D1'), ('m8an', 'S') ]
100 , 'ampn_2' : [ ('m6an_bn', 'D2'), ('m8bn', 'S') ]
101 , 'ampn_4' : [ ('m3an_bn', 'D2'), ('m6an_bn', 'G'), ('m8bn', 'D') ]
102 , 'ampn_61' : [ ('m10an', 'D'), ('m3an_bn', 'S') ]
103 , 'ampn_62' : [ ('m5an', 'D'), ('m7an', 'S') ]
104 , 'ampn_63' : [ ('m5bn', 'D'), ('m7bn', 'S') ]
105 , 'ampn_71' : [ ('m3an_bn', 'G1'), ('m4an', 'S'), ('m7an', 'D') ]
106 , 'ampn_72' : [ ('m3an_bn', 'G2'), ('m4bn', 'S'), ('m7bn', 'D') ]
107 # AmpLin P.
108 , 'ampp_1' : [ ('m6ap_bp', 'D1'), ('m8ap', 'S') ]
109 , 'ampp_2' : [ ('m6ap_bp', 'D2'), ('m8bp', 'S') ]
110 , 'ampp_4' : [ ('m3ap_bp', 'D2'), ('m6ap_bp', 'G'), ('m8bp', 'D') ]
111 , 'ampp_61' : [ ('m10ap', 'D'), ('m3ap_bp', 'S') ]
112 , 'ampp_62' : [ ('m5ap', 'D'), ('m7ap', 'S') ]
113 , 'ampp_63' : [ ('m5bp', 'D'), ('m7bp', 'S') ]
114 , 'ampp_71' : [ ('m3ap_bp', 'G1'), ('m4ap', 'S'), ('m7ap', 'D') ]
115 , 'ampp_73' : [ ('m3ap_bp', 'G2'), ('m4bp', 'S'), ('m7bp', 'D') ]
116 # Interface of internal signals to whole design.
117 , 'vcmbfb' : [ ('m1lan', 'D1'), ('m1lap', 'D2'), ('m13', 'D'), ('m9ap_an', 'G') ]
118 , 'm2n_in' : [ ('m2n', 'G'), ('m3an_bn', 'D1'), ('m8an', 'D') ]
119 , 'm2p_in' : [ ('m2p', 'G'), ('m3ap_bp', 'D1'), ('m8ap', 'D') ]
120 , 'm4ap_in' : [ ('m4ap', 'G'), ('m1ap_an', 'D1'), ('m2p', 'S') ]
121 , 'm4an_in' : [ ('m4an', 'G'), ('m1ap_an', 'D2'), ('m2n', 'S') ]
122 # Blocks external polarization.
123 , 'vref' : [ ('m1lan', 'G1'), ('m1lap', 'G2') ]
124 # vc becomes internal with vcp and vcn
125 , 'vcp' : [ ('m4bp', 'G'), ('m4bpbias', 'G'), ('m4bpbias', 'S'), ('m4bpb0', 'G')
126 , ('m4bpb0', 'D'), ('m4bpb1', 'D'), ('m4bpb2', 'D'), ('m4bpb3', 'D')
127 , ('m4bpb4', 'D'), ('m4bpb5', 'D'), ('m4bpb6', 'D'), ('m4bpb7', 'D') ]
128 , 'vcn' : [ ('m4bn', 'G'), ('m4bnbias', 'G'), ('m4bnbias', 'S'), ('m4bnb0', 'G')
129 , ('m4bnb0', 'D'), ('m4bnb1', 'D'), ('m4bnb2', 'D'), ('m4bnb3', 'D')
130 , ('m4bnb4', 'D'), ('m4bnb5', 'D'), ('m4bnb6', 'D'), ('m4bnb7', 'D') ]
131 , 'vb5' : [ ('m5an', 'G'), ('m5bn', 'G'), ('m5ap', 'G'), ('m5bp', 'G') ]
132 , 'vb7' : [ ('m7an', 'G'), ('m7bn', 'G'), ('m8an', 'G'), ('m8bn', 'G')
133 , ('m7ap', 'G'), ('m7bp', 'G'), ('m8ap', 'G'), ('m8bp', 'G') ]
134 , 'vb10an' : [ ('m10an', 'G'), ]
135 , 'vb10ap' : [ ('m10ap', 'G'), ]
136 , 'vb12an' : [ ('m12ap_an', 'G2'), ]
137 , 'vb12ap' : [ ('m12ap_an', 'G1'), ]
138 , 'vb13' : [ ('m13', 'G'), ]
139 , 'vdd' : [ ('m9ap_an', 'S'), ('m1lan', 'D2'), ('m1lap', 'D1'), ('m13', 'S')
140 , ('m5an', 'S'), ('m5bn', 'S'), ('m6an_bn', 'S'), ('m5ap', 'S')
141 , ('m5bp', 'S'), ('m6ap_bp', 'S')
142 , ('m4bpb0', 'S'), ('m4bpb1', 'S'), ('m4bpb2', 'S'), ('m4bpb3', 'S')
143 , ('m4bpb4', 'S'), ('m4bpb5', 'S'), ('m4bpb6', 'S'), ('m4bpb7', 'S')
144 , ('m4bnb0', 'S'), ('m4bnb1', 'S'), ('m4bnb2', 'S'), ('m4bnb3', 'S')
145 , ('m4bnb4', 'S'), ('m4bnb5', 'S'), ('m4bnb6', 'S'), ('m4bnb7', 'S') ]
146 , 'vss' : [ ('m1ap_an', 'S'), ('m2p', 'B'), ('m2n', 'B'), ('m1lan', 'B')
147 , ('m1lap', 'B'), ('m12ap_an', 'S'), ('m10an', 'S'), ('m4an', 'D')
148 , ('m4bn', 'D'), ('m10ap', 'S'), ('m4ap', 'D'), ('m4bp', 'D')
149 , ('m4bpbias', 'D'), ('m4bnbias', 'D') ]
150 # Transistors for digital control of biasing VC

```

A.2. SCRIPT DE LA TRANSCONDUCTANCE DIFFÉRENTIELLE CONTRÔLABLE

```
151 , 'vc1' : [ ('m4bpb1' , 'G' ), ('m4bnb1' , 'G' ), ('decoder' , 'vc(1)' ) ]
152 , 'vc2' : [ ('m4bpb2' , 'G' ), ('m4bnb2' , 'G' ), ('decoder' , 'vc(2)' ) ]
153 , 'vc3' : [ ('m4bpb3' , 'G' ), ('m4bnb3' , 'G' ), ('decoder' , 'vc(3)' ) ]
154 , 'vc4' : [ ('m4bpb4' , 'G' ), ('m4bnb4' , 'G' ), ('decoder' , 'vc(4)' ) ]
155 , 'vc5' : [ ('m4bpb5' , 'G' ), ('m4bnb5' , 'G' ), ('decoder' , 'vc(5)' ) ]
156 , 'vc6' : [ ('m4bpb6' , 'G' ), ('m4bnb6' , 'G' ), ('decoder' , 'vc(6)' ) ]
157 , 'vc7' : [ ('m4bpb7' , 'G' ), ('m4bnb7' , 'G' ), ('decoder' , 'vc(7)' ) ]
158 # Encoded command.
159 , 'cmd0' : [ ('decoder' , 'command(0)' ) ]
160 , 'cmd1' : [ ('decoder' , 'command(1)' ) ]
161 , 'cmd2' : [ ('decoder' , 'command(2)' ) ]
162 }
```

Code A.4 – Script Python de la transconductance différentielle contrôlable - Description du *slicing tree*

```
1 self.beginCell( 'gmchamla' )
2 self.doDevices()
3 self.doNets ()
4
5 self.beginSlicingTree ()
6
7 self.pushHNode( Center )
8 self.addHRail( self.getNet( 'vss' ), 'METAL4', 2, "CHI", "IH1" )
9
10 self.pushVNode( Center )
11
12 self.addSymmetryNet( VNode, self.getNet( 'vb5' ) )
13 self.addSymmetryNet( VNode, self.getNet( 'vb7' ) )
14 self.addSymmetryNet( VNode, self.getNet( 'vcp' ), self.getNet( 'vcn' ) )
15 self.addSymmetryNet( VNode, self.getNet( 'ampp_73' ), self.getNet( 'ampn_72' ) )
16 self.addSymmetryNet( VNode, self.getNet( 'ampp_71' ), self.getNet( 'ampn_71' ) )
17 self.addSymmetryNet( VNode, self.getNet( 'ampp_61' ), self.getNet( 'ampn_61' ) )
18 self.addSymmetryNet( VNode, self.getNet( 'ampp_63' ), self.getNet( 'ampn_63' ) )
19 self.addSymmetryNet( VNode, self.getNet( 'ampp_4' ), self.getNet( 'ampn_4' ) )
20 self.addSymmetryNet( VNode, self.getNet( 'ampp_2' ), self.getNet( 'ampn_2' ) )
21 self.addSymmetryNet( VNode, self.getNet( 'ampp_1' ), self.getNet( 'ampn_1' ) )
22 self.addSymmetryNet( VNode, self.getNet( 'm2p_in' ), self.getNet( 'm2n_in' ) )
23
24 self.addVRail( self.getNet( 'vdd' ), 'METAL3', 2, "CV1", "CV1" )
25 # Ampli P.
26 self.pushHNode( Center )
27 self.addDevice( 'm10ap' , Center, span=(4.0, 2.0, 2.0) )
28 self.addDevice( 'm3ap_bp' , Center, span=(2.0, 2.0, 2.0) )
29
30 self.pushVNode( Center )
31 self.addDevice( 'm4bp' , Center, span=(4.0, 2.0, 2.0) )
32 self.addDevice( 'm4ap' , Center, span=(4.0, 2.0, 2.0) )
33 self.addSymmetry( 0, 1 )
34 self.popNode()
35
36 self.pushVNode( Center )
37 self.addDevice( 'm7ap' , Center, span=(4.0, 2.0, 2.0) )
38 self.addDevice( 'm8ap' , Center, span=(4.0, 2.0, 2.0) )
39 self.addDevice( 'm8bp' , Center, span=(4.0, 2.0, 2.0) )
40 self.addDevice( 'm7bp' , Center, span=(4.0, 2.0, 2.0) )
41 self.addSymmetry( 0, 3 )
42 self.addSymmetry( 1, 2 )
43 self.popNode()
44
45 self.pushVNode( Center )
46 self.addDevice( 'm5ap' , Center, span=(4.0, 2.0, 2.0) )
47 self.addDevice( 'm6ap_bp' , Center, span=(4.0, 2.0, 2.0) )
48 self.addDevice( 'm5bp' , Center, span=(4.0, 2.0, 2.0) )
49 self.addSymmetry( 0, 2 )
50 self.popNode()
51
52 # Digitally controlled biasing
53 self.pushVNode( Center )
54 self.addDevice( 'm4bpbias' , Center, span=(2.0, 1.0, 2.0) )
55 self.addDevice( 'm4bpb0' , Center, span=(1.0, 1.0, 1.0) )
56 self.addDevice( 'm4bpb1' , Center, span=(1.0, 1.0, 1.0) )
57 self.addDevice( 'm4bpb2' , Center, span=(1.0, 1.0, 1.0) )
58 self.addDevice( 'm4bpb3' , Center, span=(1.0, 1.0, 1.0) )
```

```

59 self.addDevice( 'm4bpb4' , Center, span=(1.0, 1.0, 1.0) )
60 self.addDevice( 'm4bpb5' , Center, span=(1.0, 1.0, 1.0) )
61 self.addDevice( 'm4bpb6' , Center, span=(1.0, 1.0, 1.0) )
62 self.addDevice( 'm4bpb7' , Center, span=(1.0, 1.0, 1.0) )
63 self.popNode()
64
65 self.popNode()
66
67 # GMD.
68 self.pushHNode( Center )
69 self.addDevice( 'm1ap_an' , Center, span=(2.0, 2.0, 2.0) )
70
71 self.pushVNode( Center )
72 self.addDevice( 'm2p' , Center, span=(4.0, 2.0, 2.0) )
73 self.addDevice( 'm2n' , Center, span=(4.0, 2.0, 2.0) )
74 self.popNode()
75
76 self.addDevice( 'm9ap_an' , Center, span=(4.0, 2.0, 2.0) )
77
78 # CMC.
79 self.addDevice( 'm12ap_an' , Center, span=(2.0, 2.0, 2.0) )
80
81 self.pushVNode( Center )
82 self.addDevice( 'm1lap' , Center, span=(2.0, 2.0, 2.0) )
83 self.addDevice( 'm1lan' , Center, span=(2.0, 2.0, 2.0) )
84 self.addSymmetry( 0, 1 )
85 self.popNode()
86
87 self.addDevice( 'm13' , Center, span=(2.0, 2.0, 2.0) )
88 self.addDevice( 'decoder' , Center )
89 self.popNode()
90
91 # Ampli N.
92 self.pushHNode( Center )
93 self.addDevice( 'm10an' , Center, span=(4.0, 2.0, 2.0) )
94 self.addDevice( 'm3an_bn' , Center, span=(2.0, 2.0, 2.0) )
95
96 self.pushVNode( Center )
97 self.addDevice( 'm4an' , Center, span=(4.0, 2.0, 2.0) )
98 self.addDevice( 'm4bn' , Center, span=(4.0, 2.0, 2.0) )
99 self.addSymmetry( 0, 1 )
100 self.popNode()
101
102 self.pushVNode( Center )
103 self.addDevice( 'm7bn' , Center, span=(4.0, 2.0, 2.0) )
104 self.addDevice( 'm8bn' , Center, span=(4.0, 2.0, 2.0) )
105 self.addDevice( 'm8an' , Center, span=(4.0, 2.0, 2.0) )
106 self.addDevice( 'm7an' , Center, span=(4.0, 2.0, 2.0) )
107 self.addSymmetry( 0, 3 )
108 self.addSymmetry( 1, 2 )
109 self.popNode()
110
111 self.pushVNode( Center )
112 self.addDevice( 'm5bn' , Center, span=(4.0, 2.0, 2.0) )
113 self.addDevice( 'm6an_bn' , Center, span=(4.0, 2.0, 2.0) )
114 self.addDevice( 'm5an' , Center, span=(4.0, 2.0, 2.0) )
115 self.addSymmetry( 0, 2 )
116 self.popNode()
117
118 # Digitally controlled biasing.
119 self.pushVNode( Center )
120 self.addDevice( 'm4bnb7' , Center, span=(1.0, 1.0, 1.0) )
121 self.addDevice( 'm4bnb6' , Center, span=(1.0, 1.0, 1.0) )
122 self.addDevice( 'm4bnb5' , Center, span=(1.0, 1.0, 1.0) )
123 self.addDevice( 'm4bnb4' , Center, span=(1.0, 1.0, 1.0) )
124 self.addDevice( 'm4bnb3' , Center, span=(1.0, 1.0, 1.0) )
125 self.addDevice( 'm4bnb2' , Center, span=(1.0, 1.0, 1.0) )
126 self.addDevice( 'm4bnb1' , Center, span=(1.0, 1.0, 1.0) )
127 self.addDevice( 'm4bnb0' , Center, span=(1.0, 1.0, 1.0) )
128 self.addDevice( 'm4bnbias' , Center, span=(2.0, 1.0, 2.0) )
129 self.popNode()
130
131 self.popNode()
132
133 self.addVRail( self.getNet( 'vdd' ), 'METAL3', 3, "CV3", "IV3" )

```

A.2. SCRIPT DE LA TRANSCONDUCTANCE DIFFÉRENTIELLE CONTRÔLABLE

```
134 self.addSymmetry( 1, 3 )
135 self.popNode()
136
137 self.addHRail( self.getNet( 'vss' ), 'METAL4', 3, "CH3", "IH3" )
138 self.popNode()
139 self.endSlicingTree ()
140 self.endCell ()
```

