53081-0

9 4 M O N 2 1 8 3

ACADEMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

SCIENCES ET TECHNIQUES DU LANGUEDOC

THESE

présentée à l'Université Montpellier II - Sciences et Techniques du Languedoc
pour obtenir le **DIPLOME DE DOCTORAT**

SPECIALITE : **ELECTRONIQUE, OPTRONIQUE ET SYSTEMES**

Formation Doctorale : ***Conception assistée des systèmes informatiques, automatiques et microélectroniques***

Ecole Doctorale : ***Sciences pour l'Ingénieur***

SYNTHESE TOPOLOGIQUE DE MACRO-CELLULES EN TECHNOLOGIE CMOS

par

MORAES Fernando

Soutenue le **4 Novembre 1994** devant le Jury composé de :

MM.	D.	AUVERGNE	Professeur Université Montpellier II	Président
	C.	PIGUET	Directeur de Département CSEM Neuchâtel	Rapporteur
	J.C.	DUFOURD	Maître de Conférences ENST Paris	Rapporteur
	G.	CATHEBRAS	Maître de Conférences Université Montpellier II	Examinateur
	C.	MASSON	Chef de Service CAO VLSI Bull Paris	Examinateur
	R.	REIS	Professeur UFRGS Porto Alegre (Brésil)	Examinateur
	A.	GREINER	Professeur Université Paris VI	Examinateur
	M.	ROBERT	Professeur Université Montpellier II	Directeur Thèse

Atelier Duplication

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

Microeletrônica - UFRGS/II
CAD: Microeletrônica
Síntese automática
Máscara: Circuitos integrados

CNPq 3.04.03.00-6

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

Nº CHAMADA	Nº R.G.:
621.38-181.4(043) M827s	30679
	DATA: 13,6,95
ORIGEM: 1 / DATA: 02 06 95	PREÇO: R$ 20.00
FUNDO: CPGCC	FORN.: CPGCC

AVANT-PROPOS

Je voudrais en premier lieu exprimer mes remerciements à Monsieur G. CAMBON, directeur du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), pour m'avoir accueillie dans son laboratoire.

Je tiens a exprimer ma profonde gratitude à les Professeurs Michel ROBERT et Daniel AUVERGNE, qui ont été les responsables de la direction de mes travaux de thèse et des sincères amis. Je remercie aussi Daniel AUVERGNE qui m'a fait l'honneur d'être le Président du jury de cette thèse.

Je remercie Monsieur Christian PIGUET, Professeur au Centre Suisse d'Électronique et de Microtechnique (CSEM), et Jean-Claude DUFOURD, Maître de Conférences à l'École Nationale Supérieure des Télécommunications (TELECOM - Paris), qui m'ont fait l'honneur d'accepter d'être rapporteurs et membres du jury de cette thèse.

Je remercie sincèrement à mon ami Ricardo REIS, Professeur à l'Université Fédérale do Rio Grande do Sul (UFRGS - Brésil), pour sa participation à mon jury de thèse. Ricardo a déjà été le responsable de mes travaux au Brésil, et depuis 88 nous travaillons ensemble dans le domaine de la synthèse automatique des circuits intégrés.

Je remercie à Christian MASSON, responsable du service CAO VLSI chez BULL, pour sa participation à mon jury de thèse et pour les importantes remarques faites au niveau des la description des algorithmes de conception des circuits intégrés. Je remercie aussi à Alain GREINER, Professeur à l'Université Paris VI, pour sa participation à mon jury de thèse.

Je remercie à Guy CATHEBRAS, Maître de Conférences au LIRMM, pour sa participation à mon jury de thèse et surtout au fait d'être en tous les moments disponible à m'aider, permettant ainsi le développement de mes travaux dans les meilleures conditions. Le monde CAO est beaucoup plus simple avec Guy !

Je remercie à Philippe COLL, collègue de thèse, l'aide apporté à la révision du manuscrit. La langue française est devenue plus simple grâce à lui ! Je tiens aussi à remercier à tous mes collègues du LIRMM, qui m'ont très bien accueillie en France, spécialement Sylvie AMAT, Fabrice MONTEIRO, Leonel TORRES, Claude COUNIL, Ghislain RUDELOU, Michel IDELVERT, Nadine AZEMARD, Michel OUSSET, Meryem MELLAH, Remi ESCASSUT et André REIS.

Je remercie à la coopération franco-brésilienne "CAPES-COFECUB", qui par la concession d'une bourse d'études a permis mon séjour en France et à la réalisation de cette thèse.

Enfin, d'immenses mercis à mes parents, mes soeurs Daniela et Beatriz, en particulier à Lidia, mon épouse, qui m'a supporté avec une compréhension et une patience infinies jusqu'au terme de cette étape.

RESUME

Les problèmes majeurs de la génération automatique du dessin des masques des circuits intégrés sont la dépendance vis-à-vis des règles de dessin et le dimensionnement correct des transistors. Les méthodes traditionnelles, telles que l'utilisation de cellules pré-caractérisées, manquent de flexibilité, car les portes des bibliothèques (en nombre limité) sont dessinées et dimensionnées (indépendamment de l'application) pour une technologie donnée.

Les méthodes de synthèse automatique du dessin des masques ont pour but de surmonter ces problèmes. Les techniques les plus couramment utilisées sont le "*gate-matrix*" et le "*linear-matrix*". L'indépendance vis-à-vis des règles de dessin est obtenue en utilisant la technique de description symbolique (dessin sous une grille unitaire), et les dimensions des transistors sont définies par le concepteur ou par un outil de dimensionnement. Nous proposons une méthode et un prototype logiciel pour la synthèse automatique des masques, en utilisant le style "*linear-matrix multi-bandes*". La description d'entrée du générateur est un fichier format *SPICE* (au niveau transistor), ce qui permet d'avoir un nombre très élevé de cellules, en particulier les portes complexes (*AOI*), et ainsi avoir une meilleure optimisation lors de la phase d'assignation technologique.

Les macro-cellules générées doivent être assemblées afin de réaliser un circuit complet. Deux contraintes supplémentaires sont ainsi imposées au générateur : malléabilité de la forme et position des broches d'entrées/sorties sur la périphérie de la macro-cellule. Les macro-cellules sont assemblées en utilisant un environnement de conception industriel.

Les contributions de ce mémoire de doctorat sont d'une part le développement d'un générateur de macro-cellules flexible ayant les caractéristiques d'indépendance aux règles de dessin et d'intégration dans un environnement de macro-cellules, et d'autre part l'étude détaillée des paramètres qui déterminent la surface occupée, les performances électriques et la puissance dissipée des macro-cellules générées automatiquement.

MOTS-CLES :
- Synthèse automatique du dessin des masques de circuits intégrés
- Algorithmes pour la compilation structurelle
- Portes complexes (AOI)
- Placement-routage
- CAO micro-électronique
- Circuit intégré spécifique (ASIC), VLSI, CMOS

ABSTRACT

The main problems of the automatic layout synthesis are the design rules dependence and the transistor sizing. The traditional layout synthesis methods, like standard-cells, are not flexible, since the cells in the libraries are designed and sized for a specific technology. In this way, the designer must change his library at each technology improvement.

The automatic layout synthesis methods overcomes these problems (design rules dependence and transistor sizing). Examples of layout styles are gate-matrix and linear-matrix. The technology independence is achieved by symbolic description (layout under an unitary grid), and the transistor sizes are defined by the designer or by a sizing tool. From these two constraints, we develop an automatic layout synthesis tool, using a linear-matrix multi-row layout style. The input description for our tool is a Spice file. This descriptions allows to define a greater number of cells (mainly AOIs gates), resulting a technology mapping with less constraints.

The generated macro-cells must be assembled in order to construct a complete circuit. Two additional constraints are then imposed to the generator : variable aspect ratio and placement of the inputs/outputs pins in the macro-cell border. The macro-cells are assembled by an industrial CAD environment.

The main contributions of this thesis are the development of a macro-cell generator (with the characteristics of technology independence and easy integration in a macro-cell environment) and the analysis of the parameters playing a role in the area, delay and power consumption.

KEY-WORDS :

- Automatic layout synthesis
- Algorithms for structural synthesis
- Complex gates (AOIs)
- Placement and routing
- CAD for microelectronics
- Application specific integrated circuit (ASIC), VLSI, CMOS

TABLE DES MATIÈRES

CHAPITRE 1

Introduction générale

L'objet de ce mémoire est l'étude des méthodes de génération automatique du dessin des masques des circuits intégrés. Nous proposerons une méthode de synthèse qui nous conduira à réaliser un prototype logiciel, ceci afin de permettre l'étude des paramètres qui influent sur l'occupation de surface de silicium, les performances temporelles et la puissance dissipée.

L'évolution rapide des technologies de fabrication des circuits intégrés a pour effet d'accélérer la conception d'outils de CAO (conception assistée par ordinateur) permettant aisément la migration technologique. L'objectif est d'aboutir à la réalisation de circuits alliant hautes performances temporelles et faible puissance dissipée, et bien évidemment en utilisant une surface de silicium minimale. Ainsi, la souplesse de la méthode de conception est un point important lors l'élaboration d'un circuit. Les méthodes de conception traditionnelles, tel que les pré-caractérisées, n'ont pas cette souplesse *temps-puissance-surface* requise. Pour cela nous développerons une méthode de conception, où le concepteur peut "modéliser" son circuit, selon les contraintes imposées.

L'organisation de ce mémoire sera la suivante :

Dans un premier temps nous présenterons la méthode de conception *descendante* pour la génération d'un circuit VLSI, du type ASIC (circuit intégré à la demande). Le point de départ de cette méthode est la fonction que le concepteur veut implanter, représentée par une description comportementale. Actuellement, des outils de synthèse logique génèrent à partir d'une description comportementale la description structurelle du circuit. La description structurelle peut avoir différents types de modules, comme les contrôleurs, les machines d'état fini, les chemins de données et les éléments de mémoire. Après la détermination du plan directeur qui optimise la surface globale de la puce, chaque module sera généré par des outils spécialisés, et l'ensemble sera connecté par un routeur global.

La méthode descendante de conception sera illustrée par la génération d'une unité arithmétique et logique. A partir d'une description comportementale (en format VHDL ou *VERILOG* [THO91] par exemple) l'outil de synthèse logique génère un schéma électrique (niveau portes). A partir du schéma électrique l'outil de synthèse structurelle génère les masques du circuit.

Nous générerons les modules composés par des portes logiques (logique aléatoire). Les méthodes de génération automatique des modules en logique aléatoire les plus couramment utilisés sont le "*standard-cell*" et le "*gate-array*". Ces deux méthodes sont des approches pré-caractérisées, basées sur l'utilisation de bibliothèques de cellules. Les styles de conception automatique de masques, ayant pour caractéristiques l'absence de bibliothèques de cellules et des performances électriques modulables en fonction de la taille des transistors, sont principalement les approches "*gate-matrix*" [LOP80] et la "*linear-matrix*" [UEH81].

Le style de conception choisi sera le "*linear-matrix*" multi-bandes. Le style "*linear-matrix*" est caractérisé par le placement des transistors en deux lignes horizontales de diffusion, avec la connexion de signaux équipotentiels entre ces lignes. Les cellules du circuit seront placées en différentes lignes de cellules, ce qui caractérise une approche multi-bandes. Dans notre méthode il n'y aura pas de canaux de routage entre les bandes. Les connexions seront réalisées entre les lignes des transistors. Cette approche sera choisie pour sa régularité et pour les faibles capacités parasites d'implantation engendrées.

Après avoir choisi le style d'implantation des masques, nous détaillerons les modules du générateur implanté. Les principales étapes de la génération des macro-cellules seront l'isolation de sous-réseaux, la génération des cellules, le positionnement et le routage. Afin d'avoir de bonnes performances temporelles, le module de génération de cellules devra trouver une solution sans cassures dans les deux lignes de transistors ('N' et 'P'), et ainsi réduire les capacités parasites de diffusion.

Le résultat de la génération de la macro-cellule sera un fichier symbolique, indépendant des règles de dessin, contenant les liaisons, les transistors et les contacts. La description symbolique sera convertie en description de masques par un outil compacteur. Nous pourrons générer des circuits contenant de la logique statique et des portes de transmission. Le nombre de transistors sera limité à environ 5000 (1250 portes équivalentes). Cette restriction sera due à l'occupation de mémoire vive par le compacteur de masques et à la complexité d'évaluation temporelle des structures contenant des milliers de transistors.

Les deux caractéristiques importantes d'une approche automatique de conception, qui n'utilise pas d'éléments de bibliothèques pré-caractérisées, sont :

- la possibilité d'utilisation systématique de portes complexes (AOIs), avec un nombre élevé de combinaisons, ce qui permet d'avoir une grande richesse d'opérations, de réduire sensiblement la surface finale du circuit et d'améliorer les performances temporelles ;
- le dimensionnement "sur mesure" des transistors.

Afin d'avoir un système de synthèse de masques complet et indépendant, le générateur développé sera intégré aux autres outils du laboratoire :

- compactage des masques [CAT90] ;
- évaluation rapide de performances temporelles [DES90] ;
- dimensionnement des transistors [AZE92].

Afin de réaliser un circuit complet, contenant entre 30.000 et 50.000 transistors, les modules générés seront intégrés dans un environnement de macro-cellules (figure 1.1). Les caractéristiques que notre générateur doit avoir seront : occupation de surface prévisible, malléabilité de la forme et fixation des entrées et des sorties sur les bords de la macro-cellule.

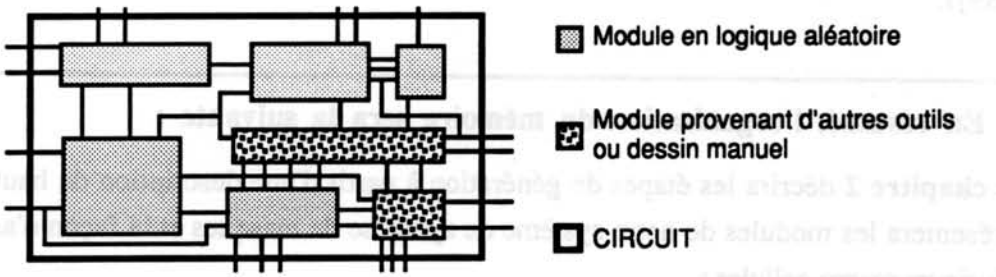


Figure 1.1 - Environnement de macro-cellules

Nous étudierons les étapes nécessaires à cette intégration :

- définition de la description d'entrée;
- partition en macro-cellules;
- création de la description interne à l'outil de CAO choisi;
- exécution du plan directeur;
- génération des macro-cellules;
- exécution du routage global.

L'évaluation de notre générateur de macro-cellules en logique aléatoire sera divisée en trois parties : analyse de surface occupée, analyse des performances électriques et comparaison avec des approches industrielles.

Les paramètres analysés dans la partie d'évaluation de surface occupée seront :

- l'influence du nombre de transistors et du nombre de bandes ;
- l'influence de l'insertion de cassures dans les connexions lors du compactage des masques;
- la réduction de la surface occupée impliquée par l'évolution de la technologie ;
- l'influence de la largeur des transistors ;
- l'influence de l'architecture choisie.

Les paramètres considérés dans la partie d'analyse des performances électriques seront :

- l'influence des capacités parasites ;
- l'influence de la largeur des transistors ;
- l'influence de la charge de sortie ;
- la réduction du retard impliquée par l'évolution de la technologie (diminution de la longueur du canal des transistors) ;
- l'effet du dimensionnement des transistors sur les performances électriques ;
- l'effet de l'utilisation des portes complexe.

Nous comparerons notre système à deux approches industrielles : *standard-cells* et *LAS*. Les paramètres de comparaison avec la méthode *standard-cells* seront la surface occupée, le retard et la puissance dissipée. L'outil *LAS* [VIR91], récemment intégré dans l'environnement CADENCE, utilise également une approche "*linear-matrix*" multi-bandes. Nous comparerons la surface occupée et le retard des deux approches en utilisant des circuits étalons (ISCAS 89 [BRG89]).

En résumé, l'organisation du mémoire sera la suivante :

- le **chapitre 2** décrira les étapes de génération à partir d'une description de haut niveau, présentera les modules de notre système de synthèse de masques et la façon d'assembler plusieurs macro-cellules ;
- le **chapitre 3** contiendra l'étude bibliographique des méthodes de synthèse de masques et la description des algorithmes implantés ;
- le **chapitre 4** détaillera les opérations nécessaires à l'assemblage de macro-cellules ;
- le **chapitre 5** présentera les résultats de surface occupée, des performances électriques et de la puissance dissipée, ainsi que la comparaison avec d'autres méthodes ;
- le **chapitre 6** présentera les conclusions de nos travaux.

CHAPITRE 2

Description du système de synthèse de masques

L'objectif de ce chapitre est de situer la méthode développée dans le contexte actuel de la synthèse des circuits VLSI. Dans les premiers paragraphes nous montrons les étapes de la génération d'un circuit à partir d'une description haut-niveau. Ensuite, nous fixons notre champ d'action, la synthèse de logique aléatoire (synthèse structurelle), ainsi que les objectifs à atteindre. La première partie du chapitre présente les outils utilisés pour la synthèse de logique aléatoire et le format des fichiers utilisés pour l'intégration de ces outils. La partie suivante illustre le mode d'intégration de plusieurs macro-cellules générées à l'aide de notre méthode, en utilisant l'environnement CADENCE. Enfin nous montrons la possibilité d'intégration de la synthèse structurelle à la synthèse logique, à partir d'une description haut-niveau.

La méthode de conception des circuits VLSI, la plus couramment utilisée, est basée sur l'édition de schémas électriques. Dans un schéma, chaque élément représente soit une cellule pré-caractérisée d'une bibliothèque soit un bloc régulier (PLA, ROM, RAM, multiplieur, etc.). Ces éléments (cellules pré-caractérisées et macro-blocs) sont ensuite placés et connectés par l'outil utilisé par le concepteur.

L'utilisation des outils de synthèse logique permet au concepteur la définition d'un circuit à partir d'un algorithme à implanter (fonctionnalité), et non plus à partir de la structure.

Les principales étapes comprises dans le processus de synthèse d'un circuit VLSI, à partir d'une description comportementale, sont décrites ci-dessous et illustrées dans la figure 2.1.

a/ Les outils de synthèse logique, comme COMPASS ou SYNOPSIS, génèrent à partir d'une description comportementale, telle que VERILOG ou VHDL, la description structurelle du circuit.

b/ Cette description structurelle est partitionnée en sous-modules, par exemple : logique aléatoire (portes logiques), machines d'état fini, chemin de données (data-path) et mémoires (RAM), qui seront traités par différents outils de synthèse, spécifiques à ces modules.

c/ Un outil de prédiction effectue l'estimation de surface et des formes possibles pour chaque bloc. Les blocs en logique aléatoire sont des blocs "*souples*", car ils peuvent changer de forme, et ainsi être facilement placés dans le plan directeur du circuit. Les blocs tels que les chemins de données ou les mémoires sont des blocs "*durs*", de faible malléabilité.

d/ En fonction des surfaces et des formes établies par l'étape antérieure, l'outil de plan directeur positionne chaque module, ayant comme fonction objectif la minimisation de la surface occupée par le circuit et la réduction du routage nécessaire à la connexion des blocs. Les premiers blocs placés sont les blocs "durs", et les espaces restants sont occupées par les blocs "souples".

e/ Ensuite chaque bloc est traité indépendamment par des générateurs spécifiques : synthèse de logique aléatoire, de machines d'état fini, de chemin de données et de mémoires. Une étape importante avant la synthèse de logique aléatoire est l'assignation technologique ("*mapping*"). La qualité de l'implémentation du module est fonction de la richesse de fonctions disponibles dans les bibliothèques spécifiées par le concepteur. L'outil de synthèse de logique aléatoire doit permettre le choix de la forme du bloc, afin de l'adapter dans le plan directeur du circuit, et permettre aussi la fixation des broches d'entrée et sortie sur les bords du bloc afin de simplifier le routage global.

f/ La dernière étape est l'assemblage des ces modules par un routeur. Le résultat est la description physique du circuit (polygones).

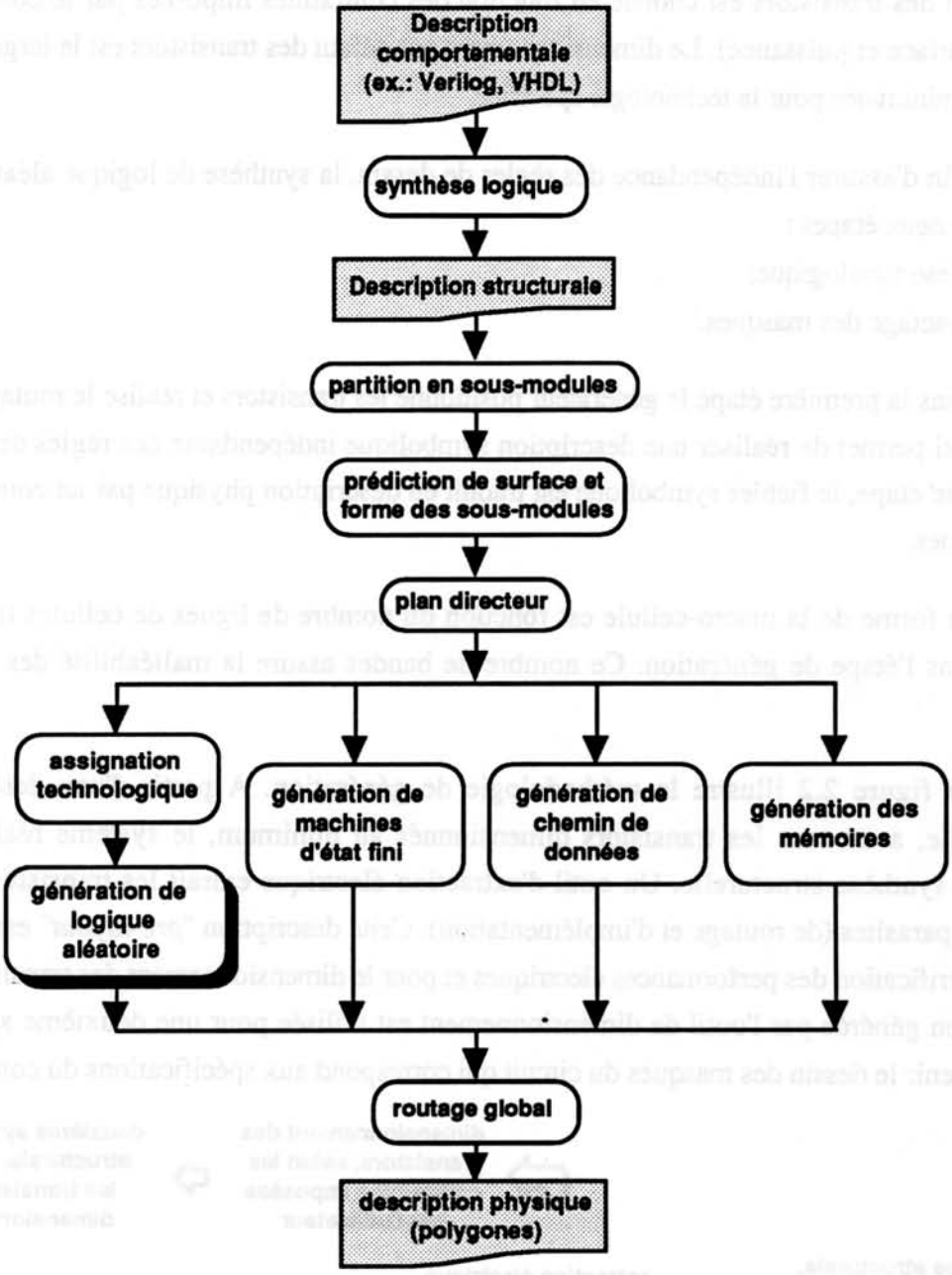


Figure 2.1 - Étapes de la génération d'un circuit à partir d'une description comportementale

Notre objectif est de réaliser la synthèse de modules en logique aléatoire, de façon à atteindre les caractéristiques suivantes:

- paramétrisation électrique,
- indépendance aux règles de dessin
- intégration à un environnement de macro-cellules (malléabilité et positionnement des broches d'entrés/sorties).

La paramétrisation électrique est obtenue par l'utilisation des bibliothèques structurelles. Dans ces bibliothèques chaque porte est définie par une liste de transistors (sous-circuit). La dimension des transistors est choisie en fonction des contraintes imposées par le concepteur (temps, surface et puissance). Le dimensionnement par défaut des transistors est la largeur et la longueur minimales pour la technologie spécifiée.

Afin d'assurer l'indépendance des règles de dessin, la synthèse de logique aléatoire est divisée en deux étapes :

- synthèse topologique;
- compactage des masques.

Dans la première étape le générateur positionne les transistors et réalise le routage entre eux, ce qui permet de réaliser une description symbolique indépendante des règles de dessin. Après cette étape, le fichier symbolique est traduit en description physique par un compacteur des masques.

La forme de la macro-cellule est fonction du nombre de lignes de cellules (*bandes*) choisi dans l'étape de génération. Ce nombre de bandes assure la malléabilité des circuits générés.

La figure 2.2 illustre la méthodologie de génération. A partir d'une description structurelle, avec tous les transistors dimensionnés au minimum, le système réalise une première synthèse structurelle. Un outil d'extraction électrique extrait les transistors et les éléments parasites (de routage et d'implémentation). Cette description "*pre-layout*" est utilisée pour la vérification des performances électriques et pour le dimensionnement des transistors. La description générée par l'outil de dimensionnement est utilisée pour une deuxième synthèse, afin d'obtenir le dessin des masques du circuit qui correspond aux spécifications du concepteur.

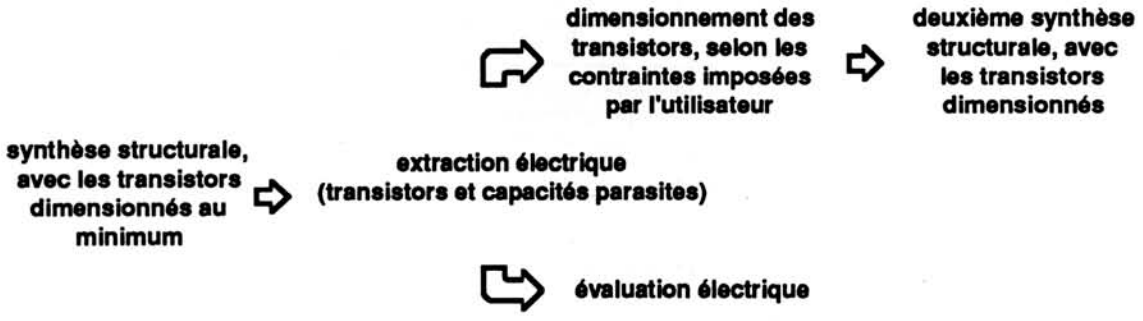


Figure 2.2 - Méthodologie de la synthèse structurelle

Nous montrerons, dans le chapitre 5, la possibilité d'obtenir une solution pour les dimensions des transistors proche de la solution dimensionnée dès la première itération. Ceci permettra d'éviter la deuxième synthèse structurelle, qui peut être très coûteuse en temps CPU selon la complexité de la macro-cellule.

2.1 OUTILS DU SYSTEME TROPIC

Nous présentons dans cette partie les outils utilisés pour la génération des macro-cellules. Notre objectif est d'obtenir une chaîne complète de conception, permettant au concepteur la réalisation d'une macro-cellule optimisée temporellement, correcte par construction et indépendant des règles de dessin. L'entrée du système est une description structurelle, soit au niveau portes, soit au niveau transistors.

Les outils nécessaires à la génération d'une macro-cellule, le format des fichiers utilisés et la séquence des opérations sont illustrés dans la figure 2.3.

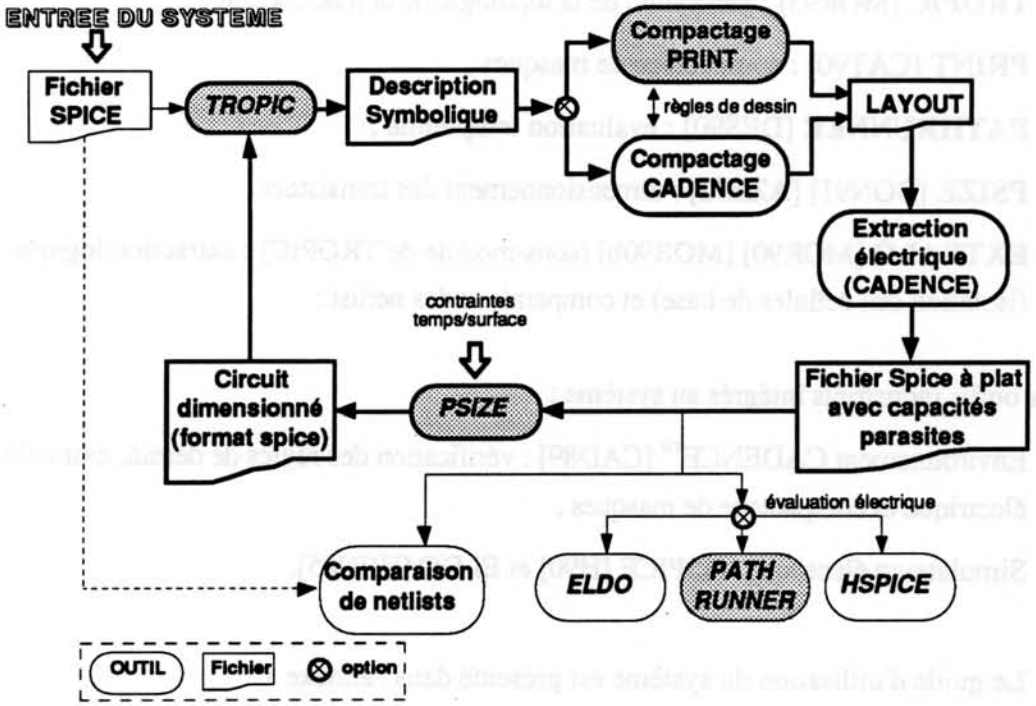


Figure 2.3 - Génération de logique aléatoire par le système TROPIC

Le générateur TROPIC génère la topologie du circuit (positionnement des transistors et routage) à partir d'un fichier de format SPICE. Le résultat de la génération est une description symbolique. Un outil de compactage réalise la conversion de cette description symbolique en une description de masques, selon les règles de dessin de la technologie choisie. Il est possible de réaliser le compactage soit avec le compacteur développé au laboratoire (PRINT), soit avec le compacteur industriel de CADENCE™.

Après le compactage de masques, l'extracteur électrique génère un fichier de format SPICE contenant les transistors, la carte modèle et les capacités parasites de diffusion et de routage. L'évaluation temporelle du circuit, la vérification de la topologie et le dimensionnement

des transistors se fait à partir de ce fichier SPICE "*post-layout*". Les simulateurs HSPICE, ELDO ou Pathrunner peuvent être utilisés pour l'évaluation temporelle. La vérification de la topologie est faite par la comparaison entre les fichiers SPICE d'entrée et "*post-layout*". Le dimensionnement des transistors est exécuté par PSIZE. Le fichier contenant les transistors dimensionnés est ensuite utilisé pour une deuxième synthèse de masques, afin d'obtenir une macro-cellule optimisée temporellement.

Cette chaîne est constituée de deux types d'outils :

- les outils développés au laboratoire :
  - **TROPIC** [MOR93] : génération de la topologie de la macro-cellule ,
  - **PRINT** [CAT90] : compactage de masques ,
  - **PATHRUNNER** [DES90] : évaluation temporelle ,
  - **PSIZE** [BON91] [AZE92] : dimensionnement des transistors ,
  - **EXTRALO** [MOR90] [MOR90b] (sous-module de TROPIC) : extraction logique (isolation des cellules de base) et comparaison des netlist ;
- les outils industriels intégrés au système :
  - Environnement CADENCE™ [CAD89] : vérification des règles de dessin, extraction électrique et compactage de masques ,
  - Simulateurs électriques HSPICE [H90] et ELDO [HEN85].

Le guide d'utilisation du système est présenté dans l'annexe 1.

2.1.1 Génération de la topologie

Afin de générer une macro-cellule optimisée temporellement, correcte par construction et indépendante des règles de dessin, nous avons adopté une démarche totalement automatique. Les cellules de base, le positionnement des transistors et le routage des signaux équipotentiels sont réalisés par notre générateur. Le style régulier d'implantation, *linear-matrix multi-bandes*, assure une haute densité d'intégration et des bonnes performances électriques.

Le développement du générateur et l'analyse de facteurs qui déterminent la surface occupée et les performances électriques d'un circuit sont les objectifs majeurs de nos travaux. Le chapitre 3 décrira l'implémentation du générateur et le chapitre 5 présentera les résultats obtenus.

2.1.2 Compactage de masques

Le circuit généré par TROPIC est décrit sous une forme symbolique, permettant de définir la topologie et la connectique du circuit, sans se soucier des règles de dessin. Cette description (contenant les liaisons, les contacts et les transistors) est traduite en une description physique par un module "compacteur". Comme il est montré dans le schéma de la figure 2.3, nous utilisons soit le compacteur PRINT [CAT90], développé au LIRMM, soit le compacteur intégré dans CADENCE™.

Le compacteur PRINT utilise deux modules:

- un *traducteur*, chargé d'assurer l'interprétation du dessin symbolique en fonction des règles de dessin de la technologie;

- un *contracteur*, outil de compactage monodirectionel, assurant la détermination des dimensions effectives des rectangles élémentaires du dessin des masques en fonction des informations élaborées par le traducteur. La détermination des dimensions du dessin de masques est effectuée par deux **contractions successives**, réalisées pour chacune des deux directions horizontale et verticale.

Le compacteur monodirectionel PRINT utilise un algorithme de compactage par ordonnancement spatial, dont la démarche est analogue à celle qui est utilisée dans une méthode de minimisation d'une fonction linéaire sous des contraintes linéaires, méthode dite "*simplexe*". Cette démarche se décompose en deux étapes:

- la détermination de l'ensemble des solutions permettant d'obtenir la cellule la plus étroite (ou la moins haute) possible;

- la sélection, dans cet ensemble, de la solution qui minimise la fonction de coût constituée par la longueur totale pondérée des connexions.

Ce système permet d'obtenir une bonne densité d'intégration, grâce au comportement prévisible du système, par opposition aux systèmes de compactage bidimensionnels, qui peuvent modifier la topologie élaborée par le concepteur.

Le compacteur industriel intégré dans l'environnement CADENCE™ (EDGE ou OPUS) est également un compacteur monodimensionel, où les dimensions des cellules sont déterminées par des contractions successives. Les différences majeures de cet outil par rapport au compacteur PRINT sont l'insertion automatique de cassures dans les fils ("*jogs*") et la facilité de migration technologique.

L'utilisation de PRINT est simple, car il peut être intégré directement au générateur TROPIC. Toutefois, l'utilisation du compacteur intégré dans CADENCE™ nécessite un module de conversion de la description symbolique vers la base de données interne à CADENCE™. Ces convertisseurs (pour EDGE et OPUS) sont présentés dans l'annexe 3.

Dans PRINT (version actuelle) la technologie est figée dans le programme. Ainsi, pour chaque changement des règles de dessin, il faut développer une "nouvelle" version du logiciel. Ce problème n'existe pas dans CADENCE™, car la technologie est décrite dans un fichier, ce qui permet aisément la migration technologique. L'annexe 4 présente la description des règles de dessin pour la technologie ECPD07 (format EDGE) et la technologie ECPD10 (format OPUS).

2.1.3 Extraction électrique

L'extraction électrique permet de déterminer les paramètres électriques liés à la technologie. Cette phase permet de reconstruire la structure réelle de la cellule, en introduisant la seconde partie de paramètres liés à la technologie, constitués par les éléments parasites relatifs à l'interaction des différents niveaux d'implantation. Cette étape est réalisée à l'aide d'un outil industriel (CADENCE™).

2.1.4 Dimensionnement des Transistors

Trois approches sont utilisées pour résoudre le problème d'optimisation globale :

- **Mathématique** [MAR89][CHE91]: le problème d'optimisation est défini par un système d'équations non linéaires sous contraintes, dans lequel les dimensions des transistors apparaissent comme autant de paramètres de conception à optimiser pour satisfaire les contraintes de vitesse ou de surface imposées. Ces méthodes, limitées à des structures de faible complexité, ont besoin d'une bonne solution initiale et ont des problèmes de convergence sur les branches de divergence;

- **Heuristique** [FIS85][HOF87]: à partir d'une solution initiale (tous les transistors dimensionnés au minimum de la technologie), une procédure itérative de dimensionnement est appliquée aux transistors jusqu'à ce que les contraintes imposées soit satisfaites;

- **Mixte** [OBE88][SHY88]: une solution initiale est obtenue à partir d'un algorithme heuristique et la solution finale à partir d'un algorithme mathématique.

Les limitations de ces méthodes sont le temps CPU élevé et les problèmes de divergence. Afin d'éviter ces problèmes, une nouvelle méthode a été développée au LIRMM: l'optimisation locale [AUV91]. Elle est basée sur l'optimisation globale de deux éléments (figure 2.4) : la cellule à dimensionner avec sa charge de sortie et l'étage de contrôle.

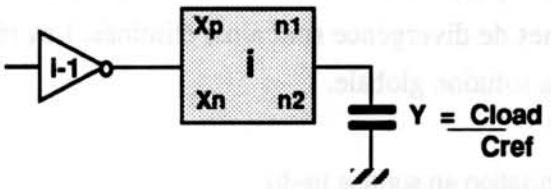


Figure 2.4 - Optimisation locale

En prenant par référence la dimension du transistor N de l'étage de contrôle, la solution de dimensionnement nécessite la détermination de trois paramètres : la dimension du transistor P de l'étage de contrôle et les dimensions des transistors N et P de la cellule considérée. La solution optimale des retards est obtenue en imposant les conditions suivantes :

- la proportionnalité entre les transistor N et P. Ceci résulte une solution avec un minimum de parasites [ROB91];
- le dimensionnement identique des transistors d'un même plan série N ou P, et l'annulation des dérivée partielles des temps de montée et de descente par rapport à la dimension du transistor N de la cellule à dimensionner;
- la symétrie des temps de montée et de descente (temps de montée = temps de descente).

Les équations résultantes sont [AUV91]:

$$X_N = \frac{\sqrt{Y^*}\,.\,(1+12K'(n_2-1))}{\sqrt{1+\frac{\mu_N}{\mu_P}+12K'\,[(n_2-1)+\frac{\mu_N}{\mu_P}(n_1-1)]}} \qquad X_P = \frac{\frac{\mu_N}{\mu_P}\sqrt{Y^*}\,.\,(1+12K'(n_1-1))}{\sqrt{1+\frac{\mu_N}{\mu_P}+12K'\,[(n_2-1)+\frac{\mu_N}{\mu_P}(n_1-1)]}}$$

où:

X_N, X_P sont respectivement les largeurs des transistors N et P, normalisées par rapport à la largeur du transistor N de référence.

Y^* est la charge normalisée, par rapport à la capacité active d'entrée du transistor N de référence.

$\frac{\mu_n}{\mu_p}$, K' caractérisent le procédé et représentent les paramètres technologiques (K' est la constant de propagation).

n_1,n_2 sont respectivement le nombre de transistors série dans les plans P et N.

Notons que pour $n_1 = n_2 = 1$, nous obtenons le dimensionnement de l'inverseur, ce qui correspond à une solution globale pour deux étages en cascade.

Ces équations permettent de dimensionner un circuit quelconque, par la superposition d'optimisations locales, en procédant des sorties vers les entrées. Les problèmes de convergence sur les branches de divergence sont ainsi éliminés. Les résultats obtenus par ces équations sont proches de la solution globale.

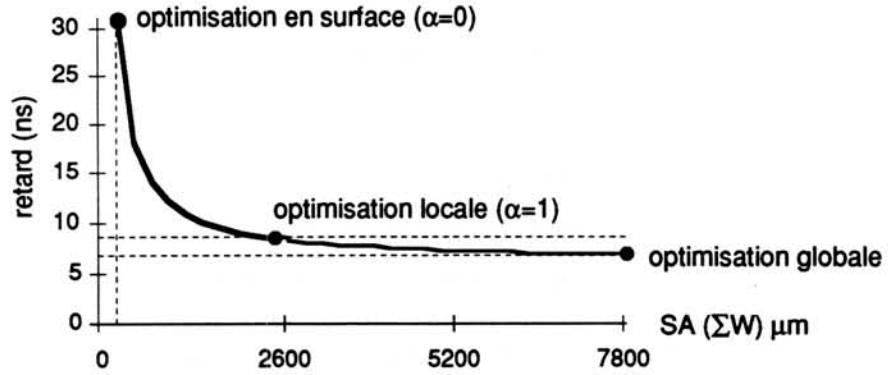


Figure 2.5 -Espace de solutions pour les dimensionnement local et global

La figure 2.5 montre l'espace des solutions de dimensionnement possibles :

- Solution avec le minimum de surface : tous les transistors ont la dimension minimale;
- Dimensionnement local : représente le meilleur compromis temps - surface;
- Solution globale : pour une solution optimale du retard, l'amélioration minime du retard par rapport à la solution local exige une forte augmentation de la surface occupée.

L'exploration de l'espace de solutions compris entre le minimum de surface occupée et la solution locale est appelée optimisation locale sous contraintes. Ceci est donné par la fonction objectif suivante:

$$\mathbf{F = \alpha.\Theta + (1 - \alpha).SA}$$

où :

α est un nombre réel compris entre 0 et 1, qui permet de fixer un poids sur les différents paramètres de la fonction objectif

Θ représente le temps critique du chemin considéré

SA représente la surface active, c'est à dire un critère de surface et de puissance dissipée sur ce chemin.

si :

$\alpha=0 \Rightarrow$ **F=SA**, détermine la solution à minimum de surface. Tous les transistors dimensionnés au minimum technologique,

$\alpha=1 \Rightarrow$ **F=Θ**, nous retrouvons la solution locale, c'est à dire, la minimisation du retard sans contraintes.

Ainsi, en tenant compte du paramètre α, les nouvelles équations de dimensionnement sont:

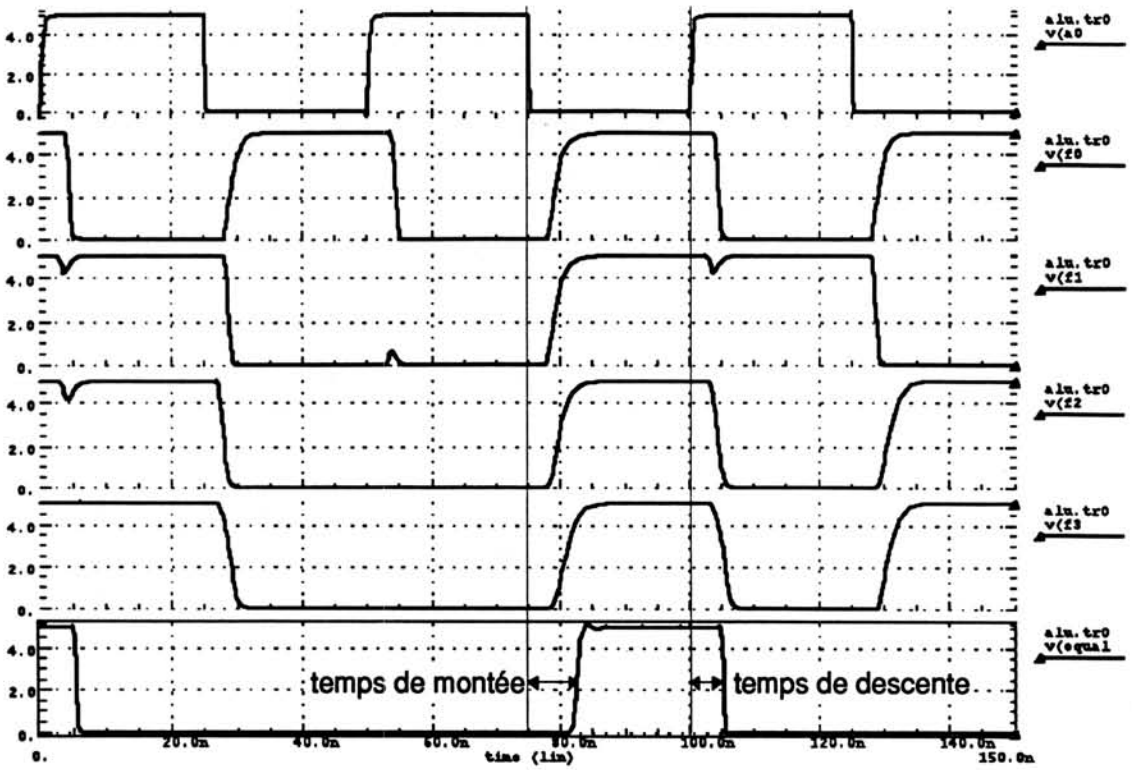
$$X_N = \frac{\sqrt{\alpha}\ \sqrt{Y^*}\ .(1+12K'(n_2-1))}{\sqrt{1+\frac{\mu_N}{\mu_P}+12K'\left[(n_2-1)+\frac{\mu_N}{\mu_P}(n_1-1)\right]}} \qquad X_P = \frac{\sqrt{\alpha}\,\frac{\mu_N}{\mu_P}\sqrt{Y^*}\ .(1+12K'(n_1-1))}{\sqrt{1+\frac{\mu_N}{\mu_P}+12K'\left[(n_2-1)+\frac{\mu_N}{\mu_P}(n_1-1)\right]}}$$

Ces équations ont été implantées dans le module informatique PSIZE [BON91]. L'entrée de cet outil est un fichier SPICE, avec les paramètres de la technologie, les transistors et les capacités parasites. Le résultat du dimensionnement est un autre fichier SPICE, où les dimensions des transistors sont fonction de l'optimisation spécifiée par le concepteur. Le compromis entre temps et surface étant imposé par le paramètre α.

2.1.5 Évaluation temporelle

L'évaluation temporelle rapide et exacte est essentielle pour l'analyse des performances temporelles d'un circuit. Nous pouvons utiliser pour l'évaluation temporelle les simulateurs *HSPICE* [H90] ou ELDO [HEN85], l'évaluateur *Pathrunner* [DES90] ou le simulateur VERILOG [THO91] (au niveau "interrupteur").

HSPICE est un simulateur électrique analogique. Les grands avantages de ce simulateur sont la précision et, pour l'utilisateur, l'interface graphique et les commandes de mesure (figure 2.6). Les commandes de mesure permettent le calcul, par exemple, du temps de montée et de descente d'un signal ou de la puissance moyenne dissipée. Ces mesures sont faites à partir du fichier de simulation, sans nécessiter la réalisation de mesures sur les courbes. L'inconvénient majeur de HSPICE est le temps CPU nécessaire à la réalisation des simulations.



(a) Interface graphique

Commandes de mesure dans le fichier de simulation:

```
.measure tran descente   trig v(a0)      val=2.5 td=20ns    rise=2
+                        targ v(equal)   val=2.5            fall=1
.measure tran montée     trig v(a0)      val=2.5 td=20ns    fall=2
+                        targ v(equal)   val=2.5            rise=1
.measure tot_power avg power    /* mesure de la puissance moyenne */
```

Mesures dans le fichier de sortie:

```
descente     = 4.8423E-09   targ= 1.0527E-07   trig= 1.0043E-07
montée       = 7.2352E-09   targ= 8.2505E-08   trig= 7.5270E-08
tot_power    = 7.7989E-03   from= 0.0000E+00   to= 1.5000E-07
```

(b) Commandes de mesure

Figure 2.6 - Simulation HSPICE

L'évaluateur temporelle *Pathrunner* est basé sur une formulation analytique des retards [DES88][AUV90], qui permet d'évaluer rapidement les performances électriques des cellules. La marge d'erreur est inférieure à 10%, sans problème de convergence. Les deux inconvénients de *Pathrunner* sont l'absence d'une interface graphique, qui rend l'interprétation des données difficile, et l'impossibilité d'utiliser des noms pour les noeuds du circuit (*Pathrunner* n'autorise que les numéros). Ceci implique un filtrage du fichier de simulation, afin de remplacer les noms par des numéros. La figure 2.7 montre le résultat de la simulation *Pathrunner* du même circuit simulé par *HSPICE*.

TIME	15	4	27	25	21	13
0.000E+00	0	1	1	1	1	1
2.132E-10	1	.	.	.	.	.
2.293E-09	.	0	.	.	.	.
2.625E-09	.	.	.	.	.	0
2.626E-09	.	.	.	0	.	.
3.912E-09	.	.	.	1	.	.
2.521E-08	0	.	.	.	.	.
2.680E-08	.	.	.	0	.	.
2.682E-08	.	.	0	.	.	.
2.707E-08	.	1	.	.	.	.
2.738E-08	.	.	.	.	0	.
5.021E-08	1	.	.	.	.	.
5.229E-08	.	0	.	.	.	.
7.521E-08	0	.	.	.	.	.
7.707E-08	.	1	.	.	.	.
7.746E-08	.	.	1	.	.	.
7.787E-08	.	.	.	1	.	.
7.841E-08	.	.	.	.	1	.
7.933E-08	.	.	.	.	.	1
1.002E-07	1	.	.	.	.	.
1.023E-07	.	0	.	.	.	.
1.026E-07	.	.	.	.	.	0
1.026E-07	.	.	.	0	.	.
1.038E-07	.	.	.	.	0	.
1.252E-07	0	.	.	.	.	.
1.268E-07	.	.	0	.	.	.
1.271E-07	.	1	.	.	.	.
1.289E-07	.	.	.	1	.	.
1.290E-07	.	.	.	.	1	.

Relation des signaux: 15=a, 4=f0, 27=f1, 25=f2, 21=f3, 13=equal

Figure 2.7 - Simulation *Pathrunner*

Le tableau ci-dessous illustre la différence entre le temps CPU pour la simulation des circuits en utilisant les simulateurs HSPICE et *Pathrunner*.

circuit	nb. de trans.	temps CPU (s) Sun Sparc 10 *HSPICE*	*Pathrunner*
dff	22	8,25	0,165
adder	28	12,17	0,265
ls161	214	138,85	0,621
alu	260	124,70	0,514
rip	448	162,47	5,743
cla	518	147,32	5,541

Tableau 2.1 - Temps CPU pour les simulateurs HSPICE et *Pathrunner* (technologie ECPD15)

Afin de résoudre le problème d'évaluation temporelle des circuits complexes, nous pouvons utiliser la simulation au niveau "interrupteur". La simulation standard au niveau "interrupteur" (Verilog [THO91]) ne permet que la définition des temps d'ouverture et de fermeture des transistors, ce qui conduit à un résultat de faible précision, n'indiquant que le fonctionnement logique de la structure. La figure 2.8 montre la description Verilog, au niveau "interrupteur", pour le circuit additionneur. La première partie (module test) contient les vecteurs de test et la deuxième partie la description de la cellule (module adder).

Cependant, le simulateur Verilog intégré dans l'environnement CADENCE permet la calibration de la technologie utilisée. Les paramètres de calibration sont les valeurs de résistances et de capacités des transistors, obtenues à partir de la simulation temporelle de

circuits de test. Les différences rapportées [CAD93] entre la simulation temporelle et la simulation au niveau "interrupteur" (calibrée) est de l'ordre de 5 %.

```
module  test;
  reg  a, b, c;
  trireg  carry, sum;

   adder top(a, b, c, carry, sum);

   initial
        begin
                $monitor ($realtime,,,,"a=%b b=%b c=%b sum=%b
                carry=%b", a, b, c, sum, carry);
                a = 1'b1;
                b = 1'b1;
                c = 1'b1;
        end
   always
                #10 a = ~ a;
   always
                #20  b = ~ b;
   always
        begin
                #40 c = ~ c;
                if( $time > 500 ) $finish ;
        end
endmodule

module  adder(a,b,c,carry,sum);
input c, b, a;
output sum, carry;
supply0 gnd;
supply1 vdd;
tranif1 #(1,1)  /* temps d'ouverture et fermeture */
        I1 (    sum, gnd,           ww1),
        I2 (    ww2,    gnd,        a),
        I3 (    ww2,    gnd,        b),
        I4 (    ww2,    gnd,        c),
        I5 (    ww1,    ww2,        ww3),
        I6 (    ww1,    ww4,        c),
        I7 (    ww4,    ww5,        b),
        I8 (    ww5,    gnd,        a),
        I9 (    ww6,    gnd,        a),
        I10 (   ww6,    gnd,        b),
        I11 (   ww3,    ww6,        c),
        I12 (   ww3,    ww7,        b),
        I13 (   ww7,    gnd,        a),
        I14 (   carry,  gnd,        ww3);
tranif0 #(1,1)
        I15 (   vdd,    sum,        ww1),
        I16 (   vdd,    ww8,        a),
        I17 (   vdd,    ww8,        b),
        I18 (   vdd,    ww8,        c),
        I19 (   ww8,    ww1,        ww3),
        I20 (   ww9,    ww1,        c),
        I21 (   ww10,   ww9,        b),
        I22 (   ww8,    ww10,       a),
        I23 (   ww12,   ww11,       a),
        I24 (   ww11,   ww3,        b),
        I25 (   ww12,   ww3,        c),
        I26 (   vdd,    ww12,       b),
        I27 (   vdd,    ww12,       a),
        I28 (   vdd,    carry,      ww3);
endmodule
```

Figure 2.8 - Exemple de description Verilog, niveau "interrupteur"

2.1.6 Comparaison des "netlists"

La comparaison des fichiers "netlists" a pour objectif de vérifier si la topologie du circuit généré correspond à la topologie initiale. Cette étape est réalisée par le module d'extraction de cellules de base du générateur TROPIC.

Les étapes de la comparaison sont décrites ci-dessous.

a/ Le comparateur construit une liste de cellules élémentaires (celles qui sont représentées par un graphe) à partir du fichier d'entrée.

b/ Étant donné que TROPIC insère les noms des signaux sur le routage, le fichier obtenu lors l'extraction électrique contient les mêmes noms de signaux que ceux du fichier d'entrée. Le comparateur construit une deuxième liste de cellules élémentaires à partir du fichier d'extraction électrique.

c/ Les deux listes sont comparées. S'il y a des différences, il signifie que le circuit généré est incorrect.

Cet outil a été développé au départ, pour la vérification du principe "*circuit correct par construction*". Dans la version actuelle du système, cette opération n'est plus nécessaire.

Le comparateur développé n'est pas un outil général, car il est basé sur la propriété d'égalité entre les noms des signaux dans les deux listes à comparer. Pour la comparaison entre netlists ayant des noms différents, il faut utiliser des algorithmes spécifiques à ce type de problème [PRE88].

2.2 ASSEMBLAGE DES MACRO-CELLULES

Après la synthèse des blocs composant le circuit, il est nécessaire de les assembler. Pour cela, nous avons développé un module qui réalise l'assemblage entre les différentes macro-cellules, afin d'obtenir un circuit complet. L'outil utilisé pour l'intégration des macro-cellules est CADENCE (OPUS ou EDGE), car il donne accès à la base de données interne au système (langage "*skill*"). Dans la version actuelle de notre méthodologie, la partition du circuit en macro-cellules et le plan directeur sont des étapes manuelles.

Ainsi, les étapes successives pour la réalisation d'un circuit complet sont les suivantes :

a/ Créer la description du circuit, soit à partir d'un schéma électrique au niveau portes soit à partir d'une description haut-niveau.

b/ Partitionner la description initiale en macro-cellules. La complexité de macro-cellules est limitée à 1250 portes (environ 5000 transistors), par les contraintes d'occupation de mémoire vive et de temps CPU nécessaire au compactage de masques. Le résultat de la partition est un fichier qui décrit l'interconnexion de macro-cellules (instances, signaux et broches d'entrée/sortie) et les descriptions SPICE (niveau portes) de chaque macro-cellule.

c/ Exécuter le plan directeur : estimation de la surface et de la forme des macro-cellules, avec l'orientation des entrées et des sorties dans ses périphéries.

d/ Générer chaque macro-cellule, la forme étant donnée par le nombre de bandes.

e/ A partir des masques des macro-cellules et du fichier qui décrit l'ensemble de macro-cellules, créer la représentation CADENCE du circuit.

f/ Placer et connecter les macro-cellules, en utilisant les outils de placement et de routage disponibles dans l'environnent CADENCE.

Cet ensemble de procédures sera détaillé au chapitre 4.

A titre d'exemple, la figure 2.9 illustre un circuit complet généré à l'aide de notre méthodologie. Le circuit est un filtre numérique [COU92], généré en 10 modules réguliers. Le nombre total de transistors est de 11300. Chaque module a été généré par TROPIC, en utilisant une solution régulière pour les dimensions des transistors.

Actuellement, notre méthodologie permet de réaliser des circuits complexes (plus de 20.000 transistors), partitionnés en macro-cellules optimisées temporellement. Les macro-cellules sont assemblées par le routeur de CADENCE. Les performances temporelles du circuit sont déterminées par la simulation au niveau "interrupteur", après calibration des paramètres liés à la technologie.

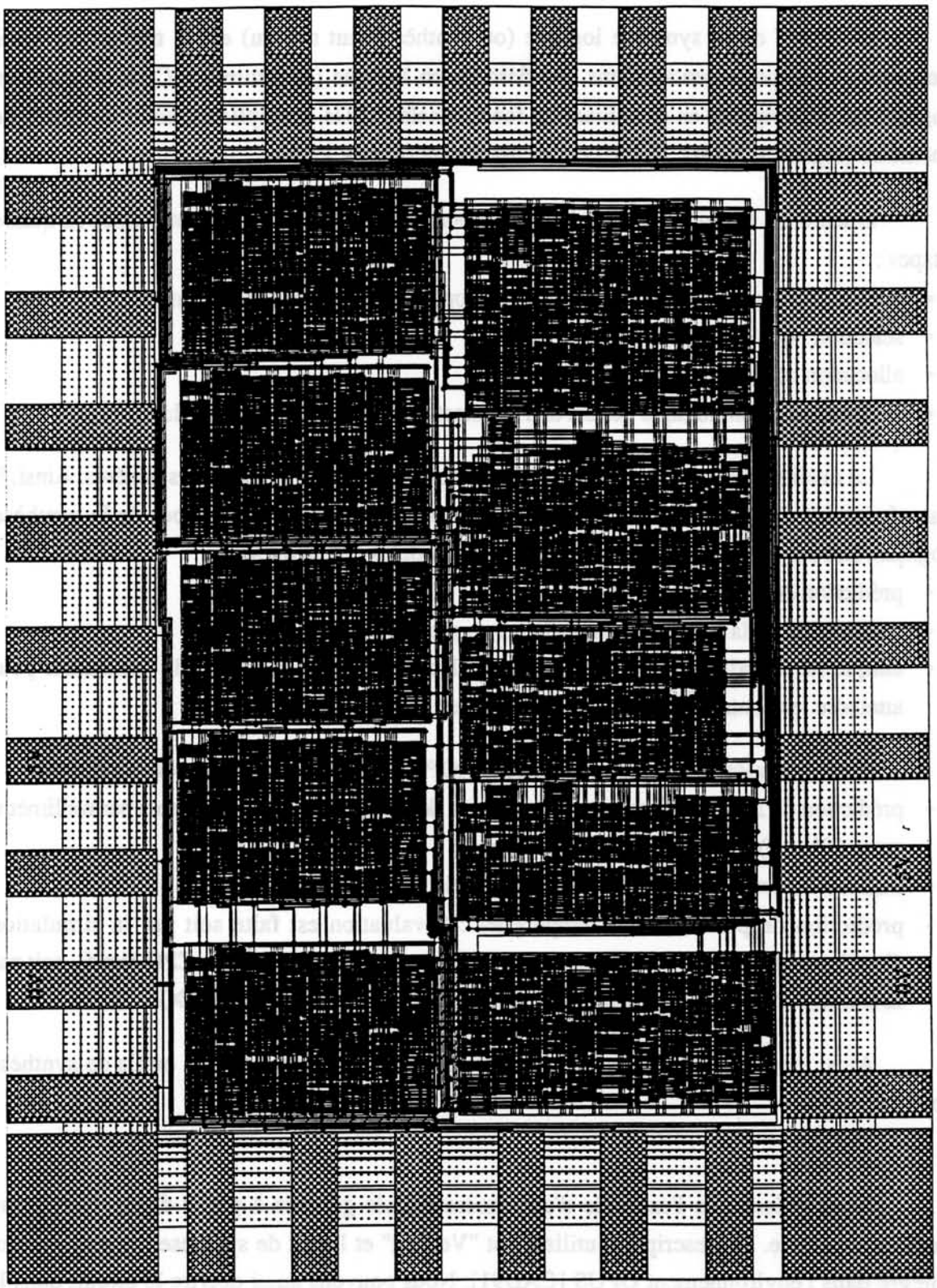


Figure 2.9 - Masques d'un circuit composé par macro-cellules (filtre numérique 19ème ordre)

2.3 INTÉGRATION AVEC LA SYNTHÈSE LOGIQUE

L'objectif de la synthèse logique (ou synthèse haut niveau) est la traduction d'une description comportementale dans une description logique (structurelle). Cette description logique est décomposée en sous-modules fonctionnels (ALUs, multiplieurs, additionneurs), de mémoire (registres, *RAMs*, *ROMs*) et de contrôleurs (portes, PLA).

L'action de traduire une description comportementale est normalement divisée en quatre étapes :

- ordonnancement ("*scheduling*") : quand les opérations doivent être exécutées;
- sélection : quels modules utiliser;
- allocation : combien de modules utiliser;
- assignation technologique : choix de l'architecture interne de chaque module.

La qualité de la synthèse logique est fonction de la qualité des modules générés. Ainsi, il est nécessaire d'ajouter des informations physiques dès les premières étapes de la synthèse logique. Les caractéristiques qu'un générateur de modules doit avoir sont :

- prédiction de la surface occupée;
- malléabilité de la forme, afin de l'adapter au plan directeur;
- simplicité d'évaluation des performances électriques, pour vérifier si le générateur peut atteindre les contraintes temporelles imposées.

Notre générateur de logique aléatoire (TROPIC) peut répondre à ces trois conditions :

- prédiction de la surface occupée: à partir de la densité moyenne de transistors par millimètre carré (T/mm^2), pour une technologie donnée;
- malléabilité : nombre de lignes de cellules variable;
- prédiction de performances temporelles : l'évaluation est faite soit par la simulation électrique en utilisant le simulateur HSPICE (le problème est le temps CPU élevé), soit par la simulation au niveau "interrupteur" en utilisant le simulateur VERILOG.

Ainsi, nous concluons que notre générateur peut être utilisé par les outils de synthèse logique.

Nous présentons l'exemple d'une macro-cellule générée à partir d'une description comportementale. La description utilisée est "Verilog" et l'outil de synthèse logique et celui intégré dans l'environnement OPUS [CAD91]. Nous pouvons aussi décrire le circuit dans le langage "VHDL", en utilisant les outils "*COMPASS*", "*SYNOPSIS*" ou "*ALLIANCE*" [GRE92]. La figure 2.10 présente la description d'une ALU (unité arithmétique et logique), en "Verilog".

```
(1)   module alu(out,en,a,b,s0,s1,s2) ;

(2)   output[7:0] out;
(3)   input[7:0] a,b;
(4)   input en,s0,s1,s2;

(5)   reg[7:0] out_i;

(6)   assign out =(en == 1)? out_i: 8'b0;

(7)   always @(s0 or s1 or s2 or a or b)
(8)       case({s0,s1,s2})
(9)           3'b100:out_i =a+b;
(10)          3'b110:out_i =a-b;
(11)          3'b111:out_i =a&b;
(12)          3'b011:out_i =a|b;
(13)          3'b001:out_i =a^b;
(14)          default:out_i = 8'bx;
(15)      endcase

(16)  endmodule
```

Commentaires de la description ci-dessus :

Ligne 1: *déclaration du module ALU, avec ses entrées et sorties.*

Lignes 2 à 4: *déclaration des signaux qui sont entrées ou sorties, et le nombre de bits par signal.*

Ligne 5: *déclaration de la variable temporaire "out_i", avec 8 bits.*

Ligne 6: *si la variable "en" est égale à 1, la sortie "out" a la valeur de la variable temporaire "out_i", sinon la sortie "out" vaut 0 (8'b0 signifie qui les 8 bits sont égaux à 0). La variable "out" est évaluée chaque fois que "en" ou "out_i" changent, car la commande "assign" a pour caractéristique d'être toujours active. Cette commande est utilisée pour la description de logique aléatoire (description de portes ou de transistors).*

Ligne 7: *la commande "always" est l'instruction de base pour décrire un processus dans Verilog. Dans l'exemple, la commande "case" est toujours exécutée lorsque une ou plusieurs des variables "s0", "s1", "s2", "a" ou "b" changent. Le caractère "@" contrôle les changements des variables.*

Lignes 8 à 15: description de l'ALU. Selon les signaux "s0", "s1" ou "s2" les opérations d'addition (a+b), de soustraction (a-b), d'et logique (a&b), d'ou logique (a|b) et d'ou exclusive (a^b) sont exécutés. Par défaut la variable temporaire "out_i" est dans une valeur indéterminé.

Ligne 16 : *fin du module.*

Figure 2.10 - Exemple de description Verilog comportementale

Cette description comportementale est ensuite utilisée directement par l'outil de synthèse logique. Le résultat de la synthèse est le schéma de la figure 2.11. Les éléments représentés par des rectangles sont des additionneurs, des soustracteurs et des multiplexeurs.

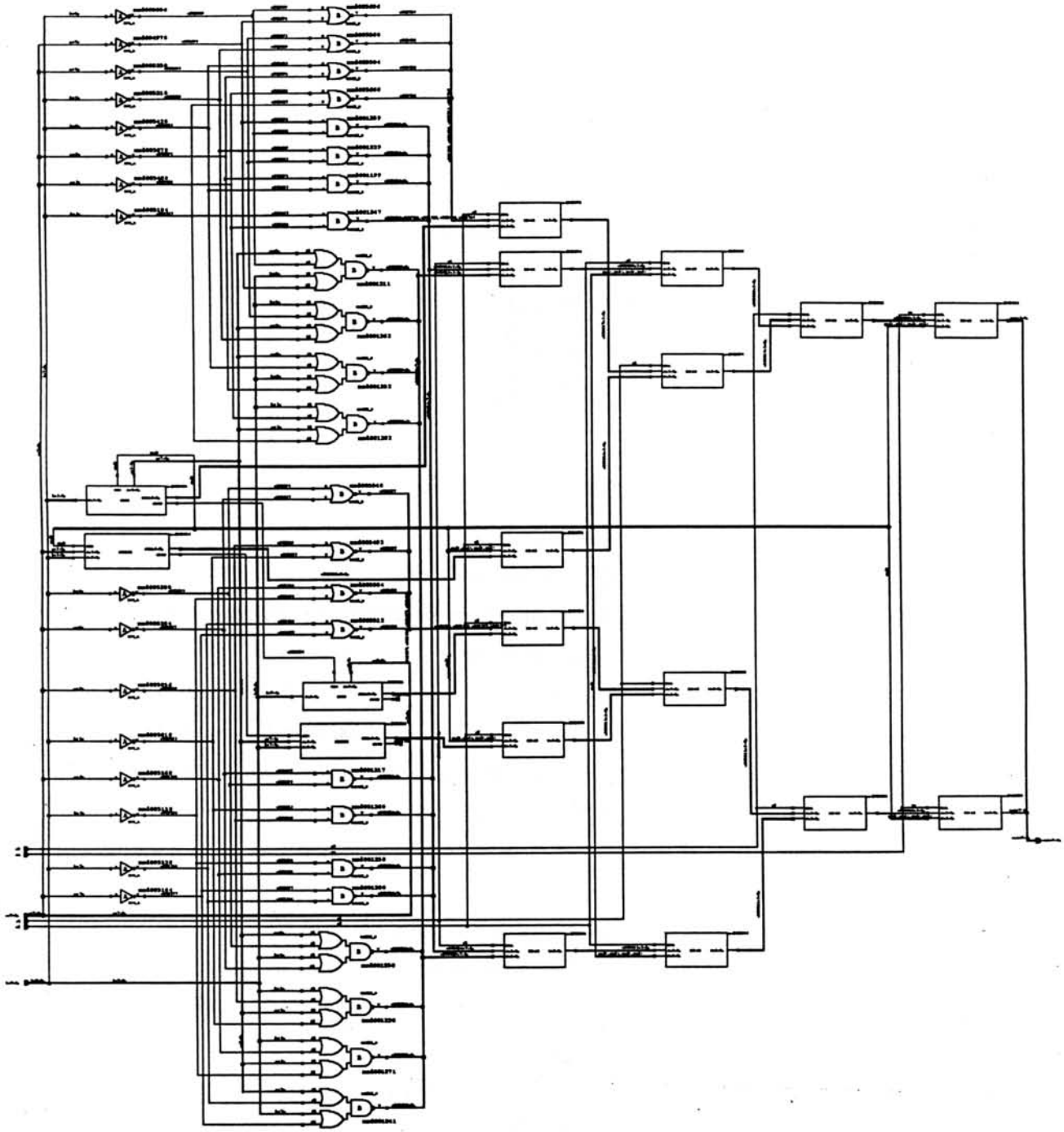


Figure 2.11 - Schéma résultant de la description comportementale de l'ALU

Nous utilisons la bibliothèque standard d'OPUS pour l'assignation technologique. Chaque porte a les représentations suivantes :

- comportementale : description de la fonction de la porte (en format Verilog);
- structurelle porte : dessin qui représente la porte (and, or, etc.);
- structurelle transistors : liste de transistors, avec les dimensions minimales pour la technologie.

Les portes peuvent également avoir les représentations :

- physique : masques de la porte ;
- physique "abstraite" : information de l'enveloppe de la cellule, position et direction des entrées/sorties. Cette information est nécessaire lors des étapes de placement et de routage de cellules.

Compte tenu du fait que les portes ont une description au niveau transistor, il est possible de générer une description SPICE à partir du schéma obtenu par la synthèse logique. Dans notre exemple, ALU, la description SPICE générée à partir du schéma contient 860 transistors (390 portes, dont 184 portes de transmission). Une fois la "netlist" SPICE générée, il suffit d'utiliser notre générateur pour obtenir les masques.

Dans cet exemple, la partition de la description structurelle en macro-cellules n'est pas nécessaire, car le circuit est de faible complexité (l'actuelle limitation est de 1250 portes ou 5000 transistors par module).

Si les portes de la bibliothèque n'ont pas de description SPICE l'utilisateur doit inclure au début de la description logique, obtenue à partir du schéma électrique, la référence à une bibliothèque structurelle (voir annexe 2). Le résultat est une description SPICE hiérarchisée.

La simulation "*pre-layout*" (sans capacités parasites) ou "*post-layout*" utilisent les mêmes vecteurs de test de la description fonctionnelle. Ceci permet de vérifier si la structure générée vérifie les contraintes temporelles imposées. Le cas échéant, il est possible soit de changer des paramètres de la synthèse logique, soit de changer la description fonctionnelle (par exemple changer l'algorithme).

L'insertion de portes complexes (AOIs) est une étape importante qui doit être ajouté au moment de la conversion de la description portes vers les transistors. [ABO92] compare le nombre de portes et de transistors nécessaires à l'implantation des circuits étalon. Il a obtenu une réduction moyenne de 32% pour les portes et pour les transistors, en utilisant des AOIs.

2.4 CONCLUSION

Ce chapitre a présenté l'objectif majeur de nos travaux : la synthèse de macro-cellules optimisées temporellement. Nous avons aussi montré la méthode d'assemblage des différentes macro-cellules afin d'obtenir un circuit complexe (plusieurs milliers de transistors), ainsi que la façon de générer un circuit à partir de sa fonction, en utilisant un outil de synthèse logique.

Les travaux futurs qui doivent être développés sont principalement :

- partition automatique d'une description logique en macro-cellules ;
- génération d'une "netlist" avec AOIs à partir d'une liste de portes ;
- détermination des vecteurs d'entrée d'un circuit complexe, afin de déterminer les performances temporelles.

CHAPITRE 3

Réalisation du Générateur de Macro-Cellules

L'objectif de ce chapitre est de montrer les principaux styles de dessins des masques de circuits intégrés et d'expliquer les méthodes utilisées pour réaliser notre générateur de macro-cellules.

Dans une première partie nous présentons les principales techniques de synthèse automatique du tracé des masques de circuits intégrés, par exemple : "gate-array", "standard-cell", générateurs de modules réguliers, "weinberger-array", "gate-matrix" et "linear-matrix". Une étude approfondie sur l'approche "linear-matrix" est effectuée, en expliquant les optimisations les plus importantes : minimisation de la largeur et de la hauteur de la cellule, et l'impact de la position des transistors dans le dessin des masques.

Cette présentation permet de justifier la structure choisie pour les modules du générateur, dont nous présentons l'organisation. Les fonctions des ces modules sont l'extraction de sous-réseaux, la génération de cellules, le partitionnement, le placement et le routage.

3.1 INTRODUCTION

Rappelons tout d'abord les principales méthodes de synthèse automatique du tracé des masques de circuits intégrés : "*gate-array*", "*standard-cell*", générateurs de modules réguliers, "*weinberger-array*", "*gate-matrix*" et "*linear-matrix*".

Réseaux pré-diffusés "*gate-array*"

Une architecture gate-array conventionnelle contient des lignes et des colonnes de cellules pré-diffusées, séparées par des canaux de routage. Les cellules sont constituées de transistors non connectés, qui par routage sur les couches de métallisation seront personnalisées pour former différents types de portes.

Les avantages de ce style de design sont la rapidité de conception et le faible coût de réalisation car seuls les niveaux de métallisation sont spécifiques du circuit. En contrepartie on peut être confronté à des problèmes de performances, de traçabilité et de dissipation.

Une variante de cette approche est le style mer de portes ("*sea-of-gates*"), où il n'y a plus de canaux de routage, les interconnexions s'effectuant au-dessus des cellules, qui sont spécifiquement conçues pour être "poreuses" au tracé sur les couches métal. Cette technique conduit a une plus grande densité d'intégration, pour les blocs réguliers.

Circuits pré-caractérisés "*standard-cells*"

Les fonctions de base (*nand*, *nor*, etc.) constituent une bibliothèque. Les cellules ont toutes la même hauteur, et sont placées en lignes horizontales, appelées bandes. Les entrées et sorties sont disposées dans les bords supérieur et inférieur de chaque cellule. Entre les bandes existe un canal de routage pour la connexion des signaux équipotentiels. Pour permettre le passage de signaux entre différentes bandes, des cellules spéciales appelées cellules de passage, sont placées entre les cellules de chaque bande (sauf si les cellules sont transparentes à la deuxième couche de métal).

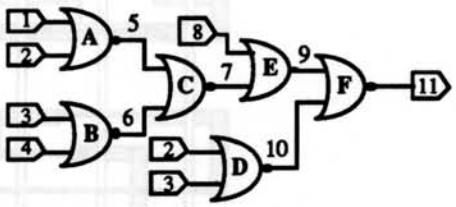
Les avantages de cette méthode sont une meilleure densité de transistors et de bonnes performances électriques. Ceci est dû à la conception manuelle des cellules de base. Outre le fait que, contrairement à un *gate-array*, les couches de diffusion sont spécifiques au circuit, la limitation majeure de cette approche est le coût de conception et de caractérisation des cellules de la bibliothèque, en raison de la dépendance vis à vis de la technologie utilisée.

Générateurs de modules réguliers

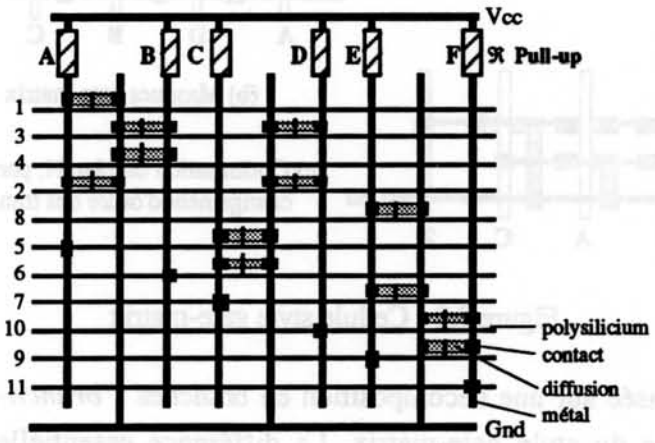
Ces générateurs sont utilisés pour des architectures qui ont une topologie facile à implanter, par la répétition d'un nombre réduit de cellules de base. Par exemple, les PLA (synthèse de logique aléatoire à deux niveaux, représentée par des sommes de produits), les mémoires (ROM et RAM), les multiplieurs ou les additionneurs.

Weinberger-array

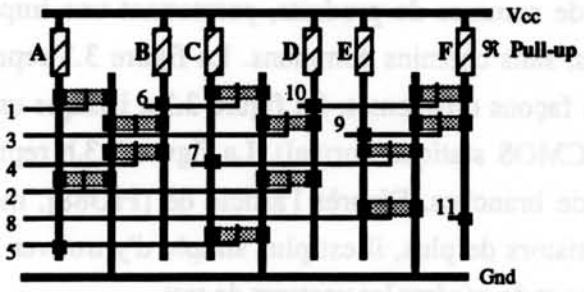
Cette approche a été la première utilisée pour la synthèse de logique multi-niveau [WEI67]. Elle utilise un ensemble de cellules *nor*, en technologie NMOS. Les transistors sont placés entre deux lignes verticales, une pour connecter les drains aux transistors de "pull-up" et l'autre pour connecter les sources à la masse. Les signaux d'entrée (grilles de transistors) et les signaux de sortie sont connectés à des rails horizontaux (figure 3.1).



(a) Schéma logique



(b) Dessin symbolique d'un circuit dans le style *Weinberger array*



(c) *Weinberger array* après optimisation (algorithme utilisé : *left-edge*)

Figure 3.1 - Cellule style weinberger-array

Gate-matrix

Cette méthode a été proposée par [LOP80], pour les cellules CMOS statiques (figure 3.2). Ce style implique des rails de polysilicium verticaux, régulièrement espacés, qui servent surtout à connecter les grilles des transistors. Les transistors sont formés par l'intersection des lignes de diffusion disposées horizontalement avec les lignes verticales de polysilicium. Le niveau métal 1 est aussi horizontal. Le nombre de colonnes de polysilicium est typiquement le nombre d'entrées/sorties du circuit, puisque les transistors ayant le même signal de grille sont placés dans la même colonne. L'approche gate-matrix a été automatisée entre autres par [WIN82][CHA87]. La principale difficulté de cette méthode est de trouver l'ordre de colonnes afin de réduire le nombre de pistes de routage nécessaires.

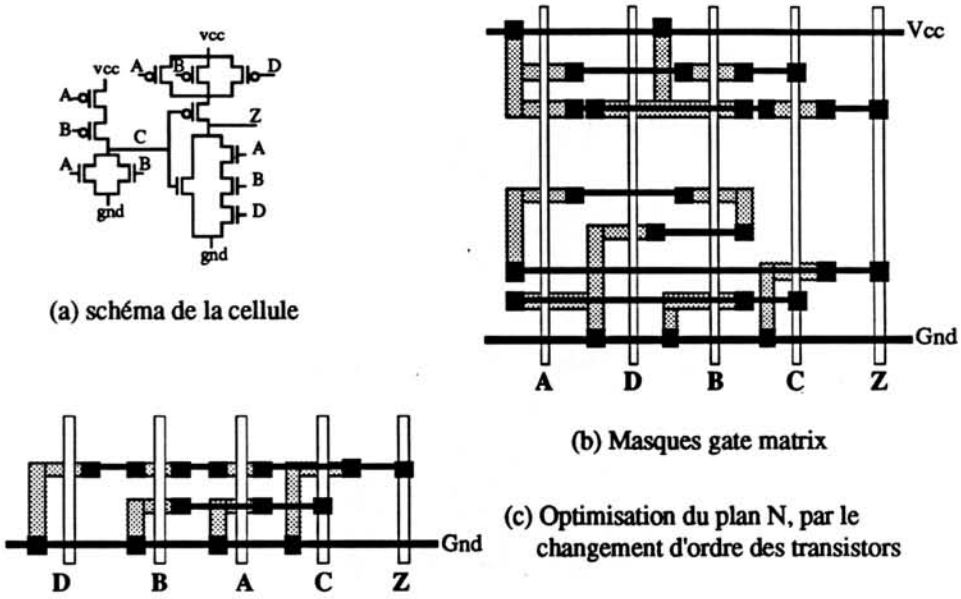
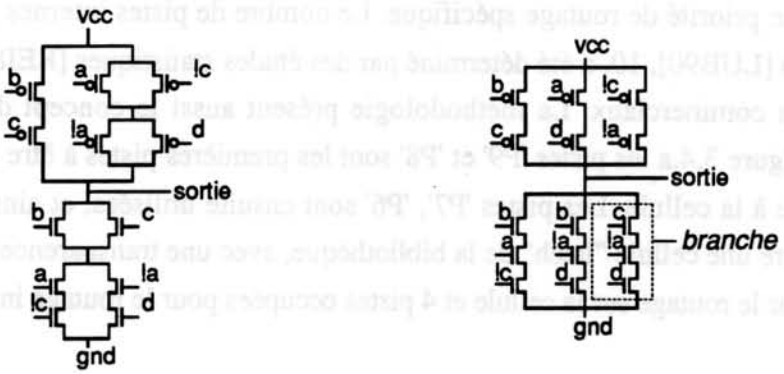


Figure 3.2 - Cellule style gate-matrix

L'approche basée sur une décomposition en branches ("*branch-based gate-matrix*" [PIG88]) est dérivée du style gate-matrix. La différence essentielle est dans le mode d'implantation des fonctions logiques. L'approche basée sur branches se limite à des équations de réseaux sous forme de sommes de produits, permettant une implantation en branches parallèles d'interrupteurs, sans chemins communs. La figure 3.3 représente la même porte logique réalisée de deux façons différentes. La figure 3.3.a indique une réalisation avec des noeuds intermédiaires (CMOS statique normal). La figure 3.3.b représente la même porte constituée uniquement de branches. D'après l'article de [PIG88], même si cette structure comporte quelques transistors de plus, il est plus simple d'y trouver le chemin critique, de dimensionner les branches et de générer les vecteurs de test.



(a) représentation CMOS statique (b) représentation sous forme de branches

Figure 3.3 - Schéma électrique d'une porte représenté sous forme de branches

Linear-matrix

Ce style de dessin de masques a été initialement proposé par [UEH81] pour la synthèse de cellules CMOS statiques. La principale différence, par rapport au gate-matrix, vient du placement de transistors. Dans cette approche il n'y a que 2 transistors ('N' et 'P') par colonne de polysilicium et les cellules sont générées en deux lignes horizontales de diffusion (aboutement linéaire des transistors de même type), si possible sans cassures de diffusion, afin d'améliorer les performances électriques de la cellule [TRA91].

L'approche basée sur une décomposition en branches peut être aussi implantée en utilisant le style *linear-matrix*. Cependant, dû au différent nombre de transistors 'N' et 'P' et aux connexions non-duales, il aura un plus grand nombre de lignes de routage, et par conséquent une surface occupée plus importante et des performances électriques compromises.

3.1.1 Les variantes du style "standard-cell"

Nous allons présenter ici 2 styles de dessin des masques dérivés de l'approche "standard-cell" traditionnelle :

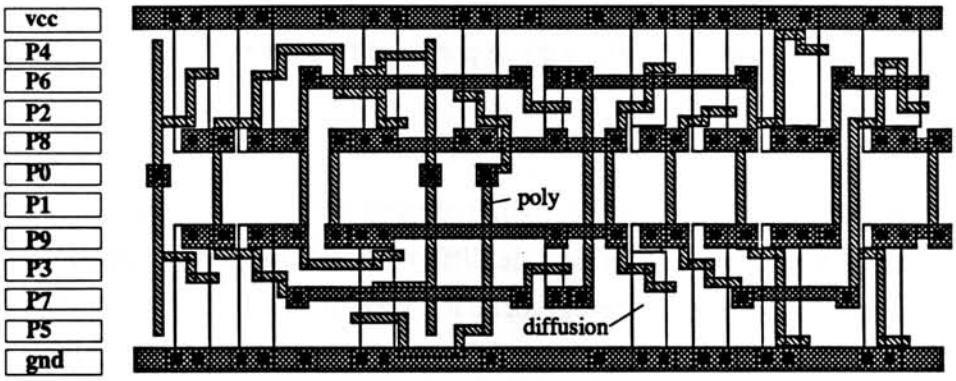
- cellules transparentes ("transparent cell approach")
- routage sur les cellules ("over-the-cell routing")

Cellules transparentes

L'objectif de cette approche est d'éliminer les canaux de routage entre les bandes [REI87][REI88].

Les cellules de la bibliothèque sont dessinées de façon à permettre le passage d'un certain nombre de pistes horizontales en métal 1 sur elles. Ce concept est appelé *transparence*. Le dessin de toutes les cellules de la bibliothèque est fait sur une grille de hauteur constante, où

chaque piste a une priorité de routage spécifique. Le nombre de pistes internes utilisé dans le système TRAMO [LUB90], 10, a été déterminé par des études statistiques [REI83], à partir de microprocesseurs commerciaux. La méthodologie présent aussi le concept de gestion des pistes. Selon la figure 3.4.a les pistes 'P9' et 'P8' sont les premières pistes à être utilisées, pour le routage interne à la cellule. Les pistes 'P7', 'P6' sont ensuite utilisées, et ainsi de suite. La figure 3.4.b montre une cellule "latch" de la bibliothèque, avec une transparence 6, c'est à dire, 6 pistes libres pour le routage sur la cellule et 4 pistes occupées pour le routage interne.



(a) priorités au routage (b) cellule *latch*

Figure 3.4 - Cellule avec transparence

Le partitionnement en bandes et le positionnement des cellules peuvent être les mêmes que ceux de l'approche "standard-cell" traditionnelle. Les algorithmes de routage sont différents [LUB90], car le routeur doit prendre en compte les contraintes d'obstacles à l'intérieur de la cellule et contrôler les pistes de routage disponibles. Le routage horizontal est fait en métal 1 et le routage vertical en métal 2.

Les avantages de cette approche sont une densité d'implémentation accrue, ainsi que de bonnes performances électriques. Les grilles des transistors étant disponibles à l'intérieur de la cellule, il n'est pas nécessaire d'y ajouter des pistes verticales pour connecter les signaux au canal externe. Une autre avantage de cette méthode est de permettre d'éviter le risque de saturation de la transparence interne des cellules, avec des bandes de hauteur variable (à réaliser dans la prochaine version du système).

L'article de [ANT92] poursuit le travail de [REI88], avec le nombre de pistes internes au routage porté à 11. Il utilise de plus un éditeur de masques spécifique à ce style de cellules avec transparence. Notons que le développement croissant de technologies à 3 niveaux de métal rend plus naturelle la notion de transparence.

Routage sur les cellules

Le but de cette approche est la réduction de la hauteur du canal de routage entre les bandes [DEU85] [CON90] [LIN91] [BHI93]. L'idée est d'utiliser un niveau de métal sur les cellules, pour le routage de quelques signaux, de façon à réduire le nombre de pistes dans le canal de routage. La figure 3.5 montre un canal de routage entre deux bandes, et quelques signaux connectés sur les cellules.

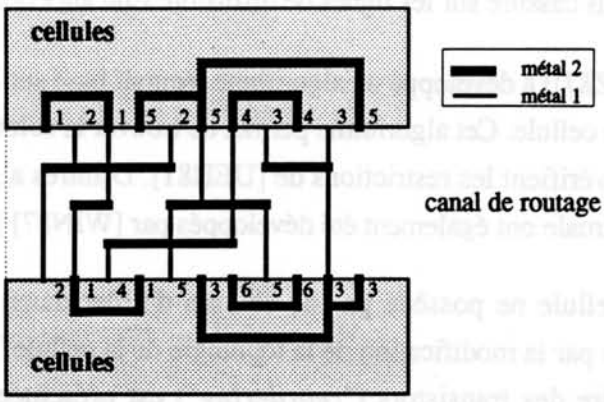


Figure 3.5 - Circuit "standard-cell", avec routage sur les cellules

La procédure pour le routage sur les cellules est la suivante :

1) ***Routage sur les cellules*** : en utilisant un niveau de métal, l'algorithme de routage essaye de connecter le plus grand nombre de signaux au-dessus des cellules. Il en résulte une réduction des connexions à effectuer dans les canaux de routage.
2) ***Choix de terminaux à connecter*** : l'algorithme doit choisir, parmi les terminaux d'un même signal qui ont été déjà connectés sur les cellules, quels sont les terminaux qui seront utilisés dans le canal de routage. Dans l'exemple, le signal '1' avait avant la première étape 4 terminaux. Après le routage sur les cellules il n'en reste que 2. Ainsi, pour réduire la longueur nécessaire pour le routage, les terminaux inférieurs et supérieurs les plus proches sont choisis.
3) ***Routage de canal*** : cette étape est exécutée pour un routeur quelconque, par exemple le routeur "greedy-router" [RIV82].

3.1.2 Les variantes du style linear-matrix

Ce style d'implémentation a été proposé par [UEH81], pour la réalisation de la synthèse automatique de masques de cellules CMOS, surtout dans le cas de cellules complexes (AOI et OAI). Par rapport à l'approche "standard-cells", une importante différence est la non-utilisation de bibliothèque des cellules. D'après l'article, les principales caractéristiques du style linear-matrix sont: cellules statiques, connexions série/parallèle, plans 'N' et 'P' duaux, deux lignes

de diffusion pour l'implémentation des transistors et transistors avec le même signal de grille alignés.

D'après [UEH81], pour minimiser la largeur de la cellule il faut connecter les transistors par aboutement. Pour cela un algorithme heuristique, basé sur l'utilisation de graphes, a été développé. L'algorithme effectue la recherche du chemin d'Euler pour chaque graphe de la cellule (plan 'N' et plan 'P'). Si le chemin d'Euler est identique pour les deux graphes, la cellule est générée sans cassure sur les lignes de diffusion. Elle aura donc la largeur optimale.

Maziaz ([MAZ87]) a développé un algorithme récursif basé sur la recherche des chemins dans les graphes de la cellule. Cet algorithme permet de trouver la solution optimale, en largeur, pour les cellules qui vérifient les restrictions de [UEH81]. D'autres algorithmes permettant de trouver la largeur optimale ont également été développés par [WIN87] [HWA89] [HWA90].

Quand une cellule ne possède pas de chemin d'Euler dans les deux graphes, il est possible d'en créer un par la modification de la topologie de la cellule. L'étude de l'influence du changement de l'ordre des transistors ("*reordering*") est présentée dans [CC88] [LEF89] [MAD89]. L'algorithme de [MAD89] accepte des contraintes de position des broches d'entrées/sortie. Ces contraintes sont utiles au moment de l'utilisation d'une cellule générée dans un environnement "standard-cell".

[CH88] et [CC89] ont élargi le domaine de la synthèse de cellules, en permettant la synthèse de logique dynamique. [CH88] présente une technique de synthèse pour les circuits ayant un nombre de transistors 'N' beaucoup plus grand que celui des transistors 'P', appelée "*transistor folding*". Elle consiste à placer la moitié des transistors 'N' proche de la masse et l'autre moitié proche de l'alimentation. Toutefois cette approche pose des problèmes lors de la réalisation du routage, résultant en espaces vides dans la cellule. Cette approche utilise des outils "standard-cell" pour le placement et le routage. Dans cette méthode, les cellules sont générées après la réalisation du placement et du routage, en tenant compte des capacités parasites. Ainsi, les cellules générées pourront donc tenir compte des contraintes imposées sur les performances. Dans le même article, les bandes générées ont une hauteur variable, égale à celle de la cellule la plus complexe de la bande. L'espace vide dans les cellules de faible complexité n'est pas utilisée.

Dans ces deux approches ([CH88] et [CC89]) les performances sont prioritaires :

- l'utilisation des couches le plus résistives (comme polysilicium) est évitée;
- la largeur des transistors devient variable;
- le nombre de trous de contacts par drain (ou source) est augmenté, car dans les approches précédentes il n'y avait qu'un seul contact par drain (ou source).

Pour ce type de génération (priorité aux performances), un système de synthèse de cellules [HSI91] traite les transistors plus larges, en les décomposant en une association de 3 transistors parallèles (figure 3.6). D'après [AMA93], la parallélisation paire d'un inverseur implique une diminution du parasite de diffusion plus importante que celle induite par une parallélisation impaire. La parallélisation impliquant la meilleure performance est le repliement par deux [AMA93].

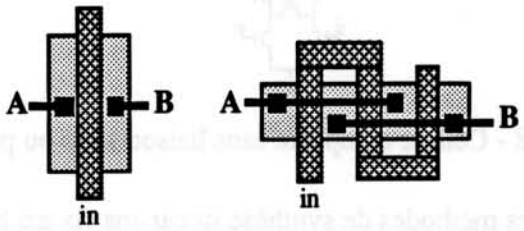


Figure 3.6 - Technique pour traiter les transistors larges

Le troisième problème à traiter dans la synthèse de cellules linear-matrix est la minimisation de la hauteur de la cellule. Dans [ONG89] la minimisation de la hauteur de la cellule est obtenue de la réduction de la densité de connexions dans le canal de routage. [MAZ91] généralise l'algorithme présenté dans [MAZ87] et développe un nouvel algorithme optimal pour résoudre les 3 problèmes : largeur, hauteur et "*reordering*". [NAK92], basé sur [MAZ91] présente aussi un autre algorithme permettant de résoudre ces trois problèmes.

La topologie de la cellule linear-matrix décomposée en 5 régions horizontales (figure 3.7) est couramment utilisée [CC89][MAZ91][HSI91][LIN92] :

- 2 régions pour les transistors;
- 2 régions pour le routage interne aux cellules (canaux de routage 'N' et 'P');
- 1 région pour le routage entre les cellules (canal de routage principal).

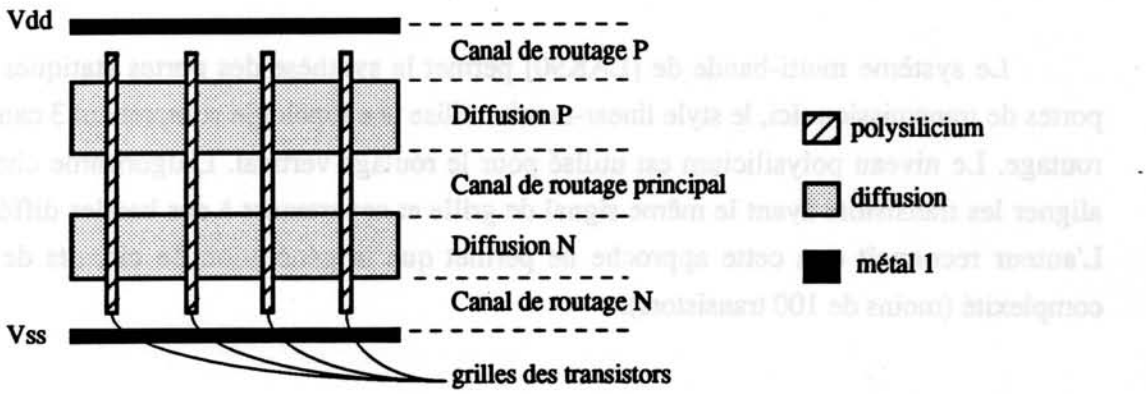


Figure 3.7 - Topologie d'une cellule linear-matrix

[CAR92] généralise le problème de la synthèse pour le cas de cellules "*planaires*" (figure 3.8), dont la particularité est de ne pas se décomposer en réseaux série/parallèle.

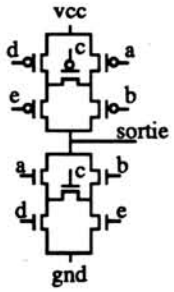


Figure 3.8 - Cellule complexe sans liaison série ou parallèle

La complexité de ces méthodes de synthèse linear-matrix est limitée à une seule cellule ou à un ensemble de cellules dans une bande (jusqu'à 200 transistors).Au-delà de cette complexité il est nécessaire d'utiliser des outils "standard-cell" (positionnement et routage) pour connecter les cellules générées.

Le système multi-bande développé par [APT87] a les caractéristiques suivantes :

- hauteur de la bande égale à la cellule la plus complexe;
- canal de routage entre les bandes;
- transparence de cellules au deuxième niveau de métal, pour permettre le routage vertical par ce niveau;
- les cellules sont gardées dans une bibliothèque. Chaque cellule contient 2 parties, 'N' et 'P', en style linear-matrix. Ces parties sont aboutées par des lignes verticales, de hauteur égale à la hauteur de la bande où la cellule est placée.

Le système multi-bande de [LAK90] permet la synthèse des portes statiques et des portes de transmission. Ici, le style linear-matrix utilise une topologie comprenant 3 canaux de routage. Le niveau polysilicium est utilisé pour le routage vertical. L'algorithme cherche à aligner les transistors ayant le même signal de grille et appartenant à des bandes différentes. L'auteur reconnaît que cette approche ne permet que la génération de circuits de faible complexité (moins de 100 transistors).

Dans [HEE92] une autre topologie multi-bande est développée, sans canaux de routage entre les bandes, avec des bandes de hauteur variable en utilisant le métal 2 pour le routage vertical. Dans ce système chaque module peut avoir jusqu'à 200 transistors. Il utilise ensuite des outils de routage pour connecter les différents modules.

Dans [HWA93] une nouvelle topologie pour le style linear-matrix est proposée (figure 3.9) :

- alimentation entre les lignes de diffusion;
- deux régions de routage;
- les sorties des cellules sont connectées en métal 2.

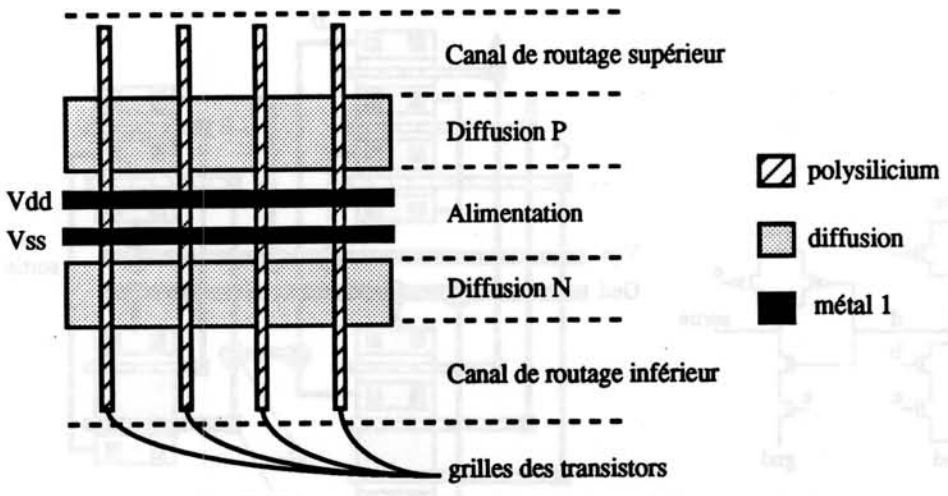


Figure 3.9 - Cellule linear-matrix avec l'alimentation entre les lignes de diffusion

Le principal avantage de cette topologie est la possibilité d'abouter toutes les cellules d'une même bande directement, sans préoccupation avec la hauteur, car les signaux d'alimentation sont placés entre les lignes de diffusion. Ce style permet également d'avoir des largeurs de transistors différentes, ceci afin de garantir les contraintes électriques imposées par le concepteur. Après l'aboutement des cellules, le canal de routage a une forme irrégulière (figure 3.10), ceci peut être pris en compte par les algorithmes de routage sur les cellules ("*over-the-cell routing*"). Ce style permet aussi de minimiser la hauteur du routage vertical, et par conséquence avoir des meilleures performances électriques.

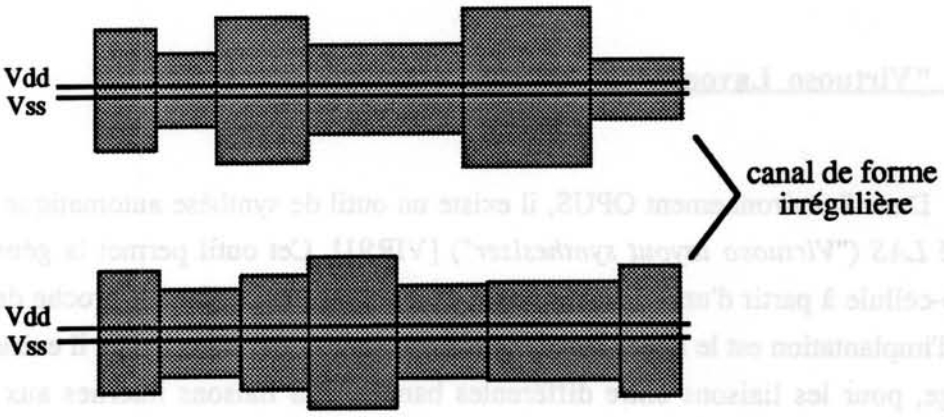


Figure 3.10 - Canal de routage pour les cellules linear-matrix avec l'alimentation entre les lignes de diffusion

Une autre topologie de layout proposée par [KIM92a][KIM92b] est basée sur le positionnement vertical de transistors, avec alimentation au milieu (figure 3.11). L'objectif de cette approche est de réduire l'utilisation du niveau polysilicium aux connexions des grilles des transistors. Le résultat est un layout avec un grand nombre de trous de contacts, une très grande surface occupée, et un grand nombre de cassures de diffusion.

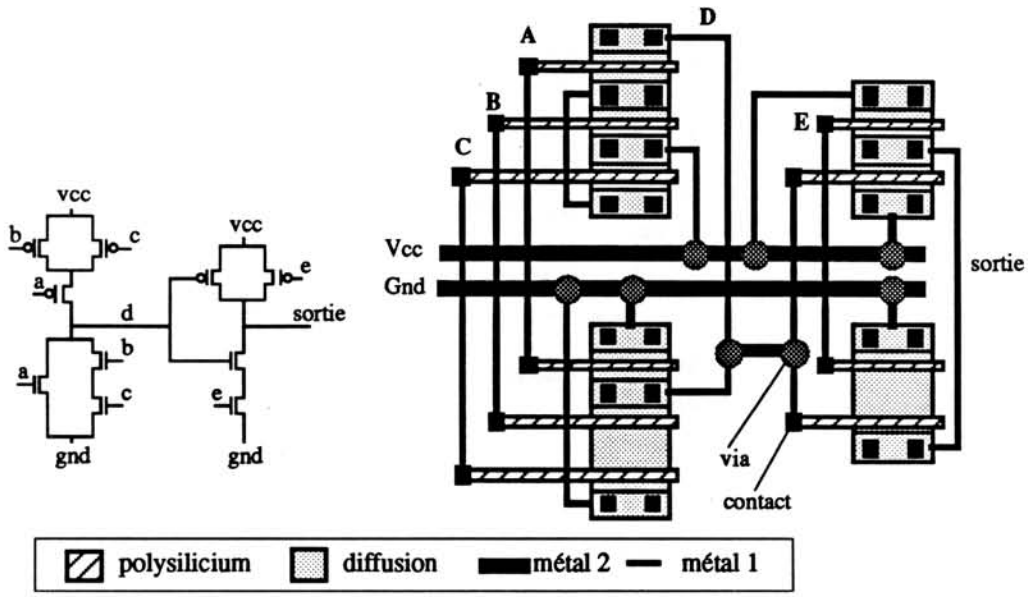


Figure 3.11 - Cellule style [KIM92a]

Il existe aussi des approches basées sur systèmes experts [KOL85][LIN87] ou systèmes hybrides [BAL88]. Les systèmes experts utilisent des règles spécifiées par le concepteur pour réaliser le dessin symbolique du circuit.

LAS- "Virtuoso Layout Synthesizer"

Dans l'environnement OPUS, il existe un outil de synthèse automatique de masques, appelé *LAS* ("*Virtuoso layout synthesizer*") [VIR91]. Cet outil permet la génération d'une macro-cellule à partir d'un schématique ou d'un fichier netlist (format proche de SPICE). Le style d'implantation est le linear-matrix, multi-bande. Entre chaque bande il existe un canal de routage, pour les liaisons entre différentes bandes. Les liaisons internes aux bandes sont réalisées entre les lignes de diffusion.

Les principales caractéristiques de *LAS* sont :

- Possibilité de choisir la position, l'ordre et la couche technologique (métal 1, métal 2 ou polysilicium) des broches d'entrée/sortie. Une broche d'entrée/sortie peut être placée sur un ou plusieurs bords de la macro-cellule.
- Possibilité de générer des circuits en logique statique ou dynamique.
- La largeur et la longueur des transistors sont variables, selon les données contenues dans le fichier d'entrée. Les transistors peuvent être parallelisés de façon automatique, à partir d'une largeur donnée.
- La forme de la macro-cellule est choisie par le nombre de bandes ou par le rapport entre sa hauteur et sa longueur ("*aspect ratio*").
- *LAS* est intégré dans l'environnement OPUS (CADENCE™). Ceci simplifie l'intégration avec d'autres outils (édition symbolique, compactage de masques, vérification des règles de dessin, extraction électrique et routage global), car la base de données est la même.

Nous présenterons l'évaluation de *LAS* dans le chapitre consacré aux résultats d'évaluation.

3.1.3 Objectifs de l'approche proposée

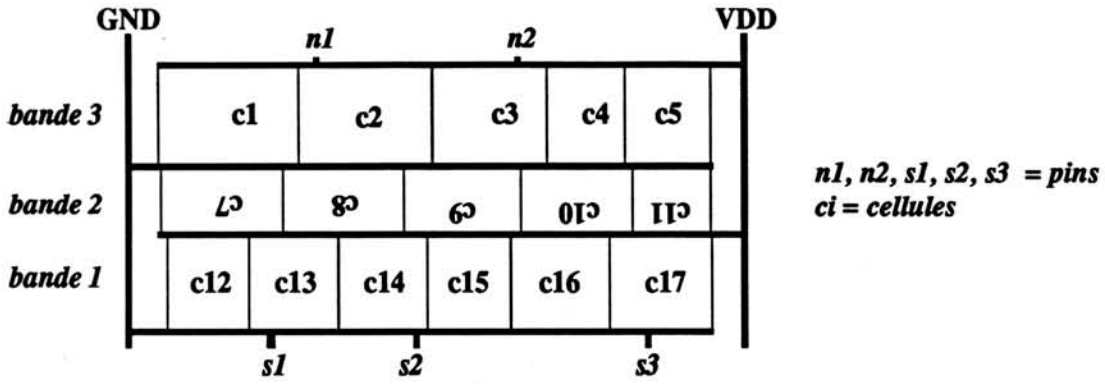
Notre principale contribution se situe dans la génération automatique du tracé de masques de macro-cellules complexes. Au vu des différents styles présentés, nous choisissons l'approche linear-matrix multi-bandes, en raison de sa régularité et de ses bonnes performances électriques. Toutes les étapes de la génération sont définies en fonction des performances : minimisation de capacités parasites dans les cellules, groupement de cellules fortement connectées et minimisation de la longueur du routage, par exemple.

Dans notre approche, les caractéristiques des masques au niveau macro-cellule sont présentées ci-dessous.

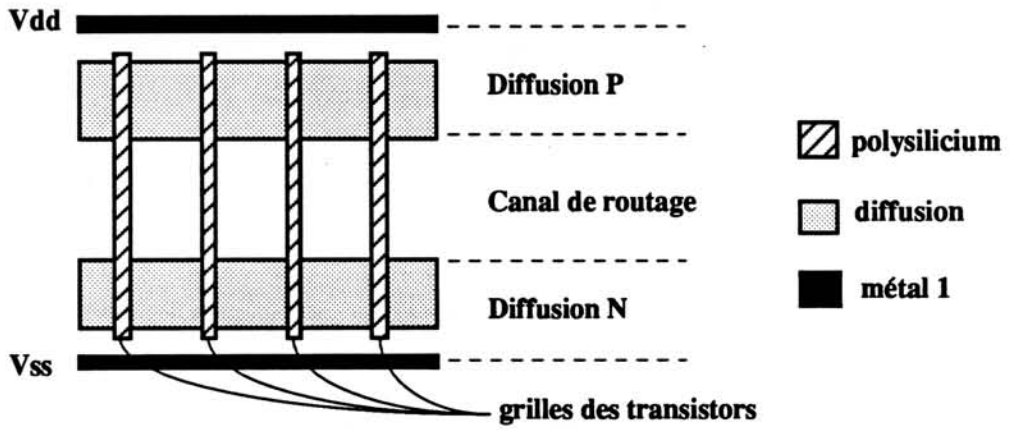
- Les cellules sont placées en lignes horizontales, ce qui caractérise un système multi-bandes.
- Il n'y a pas de canaux de routage entre les bandes. La hauteur de chaque bande est proportionnelle au nombre de pistes de routage nécessaires pour connecter tous les signaux de la bande.
- Les bandes sont aboutées par les lignes d'alimentation. Pour cela, les bandes de nombre pair sont inversées, par rapport à l'axe horizontal des alimentations.

- Il n'y a pas de cellules de passage. Les cellules générées sont transparentes au deuxième niveau de métal.
- Le générateur accepte des contraintes de positionnement des broches d'entrée ou sortie, sur les bords de la macro-cellule.
- Chaque macro-cellule peut avoir jusqu'à 5000 transistors. ***Cette complexité est fonction des limitations imposées par les outils de compactage de masques.***
- L'alimentation est distribuée par un peigne en métal 1, entre les bandes.

La figure 3.12 montre la topologie des macro-cellules et des cellules générées à l'aide du logiciel réalisé.



(a) niveau macro-cellule



(b) niveau cellule

Figure 3.12 - Topologie des macro-cellules et des cellules

Les caractéristiques des masques au niveau cellule sont présentées ci-dessous.

- Le générateur utilise des cellules en logique CMOS statique et des portes de transmission.
- Les cellules sont en style linear-matrix, avec un canal de routage. Ce canal est utilisé pour les connexions entre différentes cellules, et pour les liaisons internes aux cellules. Ce canal est divisé en trois régions : région 'N', région centrale et région 'P'. Les régions 'N' et 'P' sont utilisées pour les liaisons internes aux cellules, et la région centrale pour les liaisons entre différentes cellules ou bandes.
- La largeur et la longueur des transistors sont variables.
- Pour réduire les éléments parasites, des contacts multiples pour le drain et la source sont utilisés.
- Le niveau diffusion n'est utilisé que pour l'implantation des transistors, et l'aboutement entre eux. Pour des raisons électriques, toute connexion en diffusion est interdite.
- Le niveau polysilicium est utilisé verticalement pour la connexion des grilles des transistors duaux. La hauteur de ces lignes est proportionnelle à la hauteur du canal de routage, ce qui implique une source importante de parasites [AMA93].
- Le niveau métal 1 est utilisé horizontalement dans le canal de routage, et verticalement pour les connexions aux alimentations.
- Le niveau métal 2 est utilisé verticalement pour relier les plans duaux (c'est à dire, connecter les noeuds de sortie des cellules) et pour le routage entre différentes bandes.

La figure 3.13 montre une macro-cellule générée par notre système, en 3 bandes, avec 260 transistors (circuit ALU 4 bits).

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

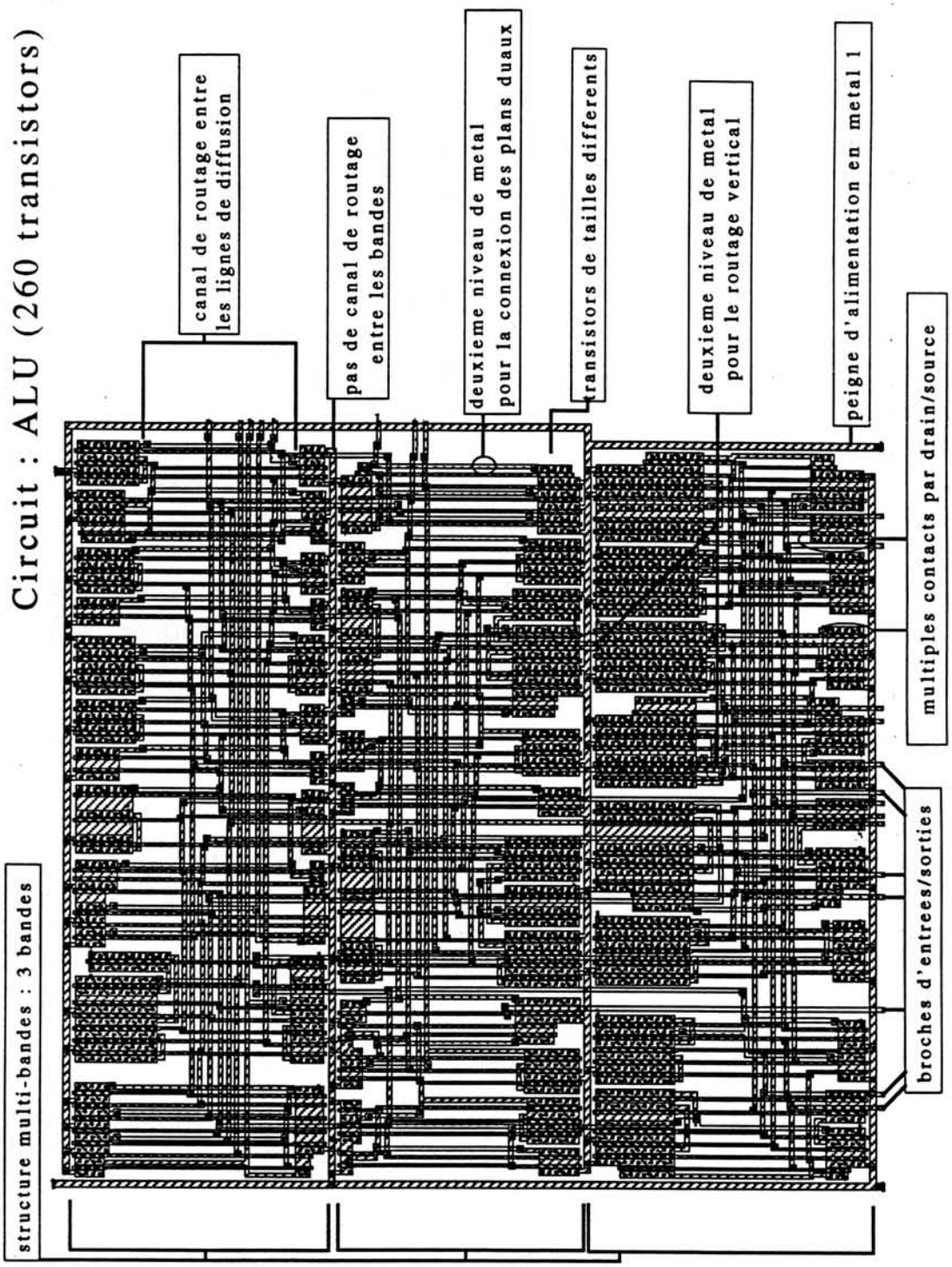


Figure 3.13 - Masques du circuit ALU 4 bits

3.2 STRUCTURE DU GENERATEUR DE MODULES

La figure 3.14 illustre les principales étapes de la génération de macro-cellules: extraction des sous-réseaux, partitionnement des cellules en bandes, génération des cellules, positionnement et routage.

Le fichier utilisé en entrée du générateur est une description structurelle, au niveau des transistors ou des portes logiques. A partir de cette description, toutes les cellules de base contenues dans le fichier d'entrée sont isolées. L'isolation des cellules utilise un algorithme de recherche de liaisons série/parallèle, l'isolation des portes de transmission est basée sur la recherche de transistors de type 'N' et 'P' en parallèle. Pour avoir une surface de silicium réduite et de meilleures performances électriques il est recommandé d'utiliser de portes complexes (AOI et OAI).

Chaque cellule est ensuite générée selon les contraintes de dimensionnement imposées par le concepteur. Aucune bibliothèque de cellules pré-caractérisées n'est nécessaire. L'algorithme implanté cherche le chemin d'Euler pour chaque cellule, en minimisant les cassures de diffusion. Le système n'accepte pas de cellules réalisées en logique dynamique (nombre de transistors 'N' différent du nombre de transistors 'P') ni de cellules avec des liaisons en topologie de pont (figure 3.8).

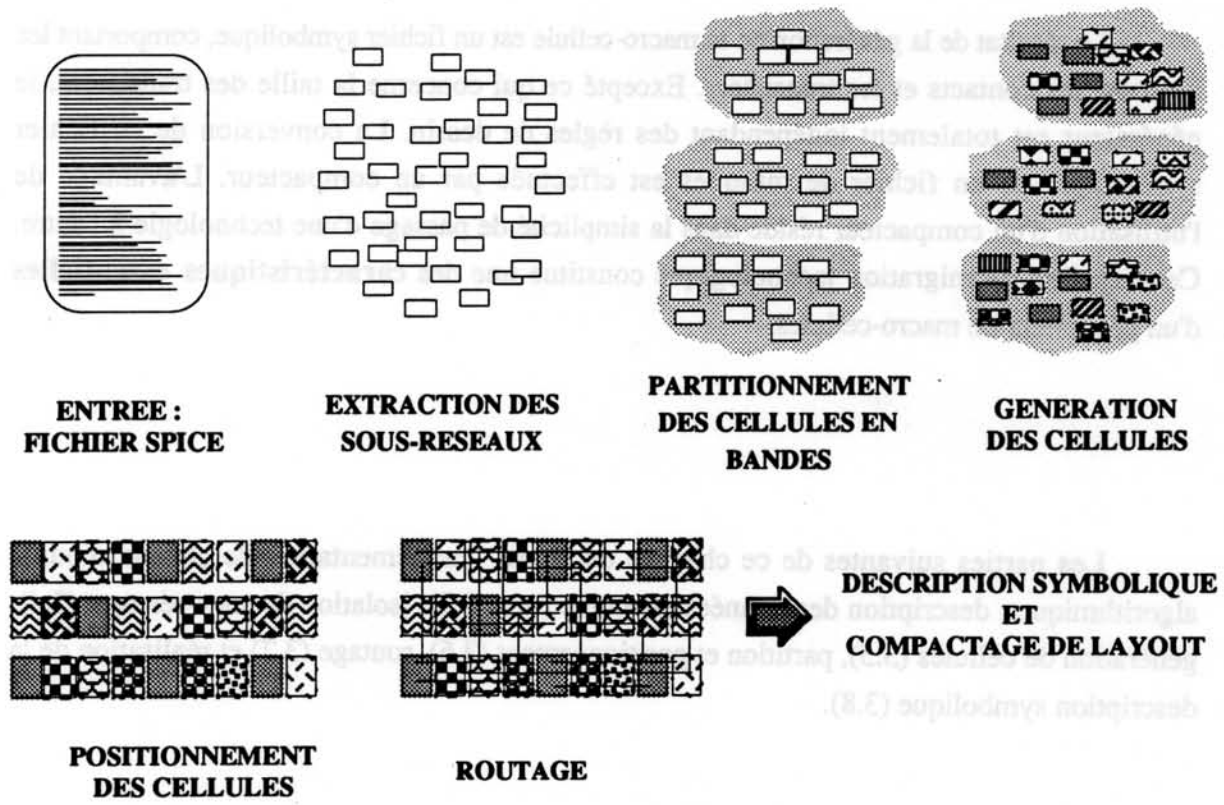


Figure 3.14 - Structure du générateur

L'algorithme de partition en bandes est composé de trois étapes :

- *Formation de groupes*, étape où les cellules le plus fortement connectées sont groupées.
- *Formation de bandes*, dans chaque bande sont placés les groupes qui minimisent le nombre de connexions externes de ces groupes. Ceci est imposé pour réduire la longueur des connexions nécessaires au routage et par conséquent améliorer les performances électriques de la macro-cellule.
- *Homogénéisation des bandes*, les cellules sont déplacées entre les différentes bandes, afin d'avoir, si possible, la même largeur des bandes. L'objectif est d'obtenir une occupation minimale de la surface.

Dans chaque bande, le positionnement des cellules est effectué afin d'obtenir le plus grand nombre d'aboutements possibles. Pour chaque bande successivement traitée, on procède à la formation d'un ensemble de solutions candidates pour le placement des cellules. Le routage horizontal (en métal 1) et vertical (en métal 2) est alors effectué pour chaque solution candidate, la solution retenue étant celle qui minimise la hauteur de la bande et la longueur des connexions nécessaires au routage.

Le résultat de la génération de la macro-cellule est un fichier symbolique, comportant les liaisons, les contacts et les transistors. Excepté ce qui concerne la taille des transistors, le générateur est totalement indépendant des règles de dessin. La conversion de ce fichier symbolique en un fichier de masques est effectuée par un compacteur. L'avantage de l'utilisation d'un compacteur réside dans la simplicité de passage d'une technologie à l'autre. Cette facilité de migration technologique constitue une des **caractéristiques essentielles** d'un générateur de macro-cellules.

Les parties suivantes de ce chapitre décrivent l'implémentation de chaque module algorithmique : description des données en entrée (item 3.3), isolation de sous-réseaux (3.4), génération de cellules (3.5), partition et positionnement (3.6), routage (3.7) et réalisation de la description symbolique (3.8).

3.3 DESCRIPTION DES DONNEES D'ENTREE

Nous avons choisi pour le format de description du fichier d'entrée un format compatible avec celui de *SPICE* [VLA80], pour 3 raisons principales:

- la description Spice est connue par tous les concepteurs de circuits intégrés,
- la description Spice est utilisée sur toutes plates-formes de CAO,
- le circuit peut être défini par une liste de transistors ou par une liste de portes.

Au niveau transistor nous utilisons une description Spice à plat, c'est à dire, une simple liste de transistors complétée de l'orientation des entrées et sorties sur les bords de la macro-cellule. Cette description est la seule source d'information nécessaire au générateur.

Au niveau portes nous utilisons une description Spice hiérarchisée. Cette description est constituée de trois parties : la définition des portes (sous-circuits), la référence de ces portes et l'orientation des entrées/sorties. Les descriptions des sous-circuits peuvent constituer une *bibliothèque structurelle*, et dans la description Spice il suffit d'inclure la bibliothèque et les références aux sous-circuits. Le résultat est une description facile à générer à partir des outils de synthèse de plus haut niveau. Par exemple, en utilisant un outil de synthèse comportemental qui génère à partir de VHDL une description au niveau portes, il est très simple de convertir n'importe quelle description dans ce niveau (portes) en une description Spice hiérarchisée.

3.3.1 Description à plat

Dans une description Spice à plat (figure 3.15) seulement 2 éléments sont lus par le module de lecture:

- les transistors,
- les contraintes de position des entrées et sorties (obligatoire).

La syntaxe des transistors est la suivante:

M<identification> <drain> <grille> <source> <substrat> <MODEL> W <valeur> L <valeur>

avec:

<identification> : une chaîne des caractères avec un numéro. Exemple : MN3, MTH4, mp2, M4.

<drain> <grille> <source> : nom des signaux connectés aux terminaux du transistor.

NB : il est préférable d'utiliser des noms et pas des numéros. La description est ainsi plus lisible.

<substrat> : noeud associé au substrat. Il sert au générateur pour identifier le nom des noeuds d'alimentation (vcc et gnd).

<MODEL> : identification du type du transistor ('N' ou 'P'). Par défaut les modèles de transistors sont 'NMOS' et 'PMOS'. Si le concepteur veut les modifier, il faut le déclarer dans la carte "*.MODEL*".

W <valeur> L <valeur> : valeur de la largeur et de la longueur du transistor. L'ordre n'est pas important, nous pouvons déclarer 'W' avant 'L' et vice-versa.

L'orientation des entrées/sorties de macro-cellules est ajoutée par l'intermédiaire d'un code de commentaire ('*') suivi par le mot réservé "interface". La syntaxe est la suivante :

**interface <nom> orientation [n, s, e, o]*

avec:

<nom> : nom du signal d'interface.

[n, s, e, o] : position du signal sur les bords de la macro-cellule - nord, sud, est, ouest.

```
* INPUT FILE - adder

M1    sum   8   vdd   vdd   PMOS   l=1.6u   w=12u
M2    7     a   13    vdd   PMOS   l=1.6u   w=12u
M3    13    b   10    vdd   PMOS   l=1.6u   w=12u
M4    10    c   8     vdd   PMOS   l=1.6u   w=12u
M5    8     16  7     vdd   PMOS   l=1.6u   w=12u
M6    7     a   vdd   vdd   PMOS   l=1.6u   w=12u
M7    vdd   b   7     vdd   PMOS   l=1.6u   w=12u
M8    7     c   vdd   vdd   PMOS   l=1.6u   w=12u
M9    vdd   16  carry vdd   PMOS   l=1.6u   w=12u
M10   15    a   vdd   vdd   PMOS   l=1.6u   w=12u
M11   vdd   b   15    vdd   PMOS   l=1.6u   w=12u
M12   15    c   16    vdd   PMOS   l=1.6u   w=12u
M13   16    b   1     vdd   PMOS   l=1.6u   w=12u
M14   1     a   15    vdd   PMOS   l=1.6u   w=12u
M15   sum   8   gnd   gnd   NMOS   l=1.6u   w=12u
M16   gnd   a   12    gnd   NMOS   l=1.6u   w=12u
M17   12    b   9     gnd   NMOS   l=1.6u   w=12u
M18   9     c   8     gnd   NMOS   l=1.6u   w=12u
M19   8     16  2     gnd   NMOS   l=1.6u   w=12u
M20   2     a   gnd   gnd   NMOS   l=1.6u   w=12u
M21   gnd   b   2     gnd   NMOS   l=1.6u   w=12u
M22   2     c   gnd   gnd   NMOS   l=1.6u   w=12u
M23   gnd   16  carry gnd   NMOS   l=1.6u   w=12u
M24   gnd   a   3     gnd   NMOS   l=1.6u   w=12u
M25   3     b   16    gnd   NMOS   l=1.6u   w=12u
M26   16    c   10    gnd   NMOS   l=1.6u   w=12u
M27   10    b   gnd   gnd   NMOS   l=1.6u   w=12u
M28   gnd   a   10    gnd   NMOS   l=1.6u   w=12u

*interface a          orientation N
*interface b          orientation N
*interface c          orientation N
*interface carry      orientation S
*interface sum        orientation S
```

Figure 3.15 -Description Spice à plat

3.3.2 Description hiérarchisée

La syntaxe de la description Spice hiérarchisée (figure 3.16) est la suivante:

• Description de sous-circuits :

.subckt <non> <liste de signaux> vcc
<transistors>
.ends <nom>

avec : *<non>* : nom du sous-circuit
<liste de signaux> : liste des signaux d'interface du sous-circuit
<transistors> : description des transistors composant la porte

• Références aux sous-circuits :

X<numéro> <liste de signaux> vcc <non>

avec : *<numéro>* : identification du sous-circut

```
* INPUT FILE - adder

.subckt gate1 a b c d e f g out vdd
MP1     5       a       vdd     vdd  PMOS  l=2e-6 w=3e-6
MP2     5       b       vdd     vdd  PMOS  l=2e-6 w=3e-6
MP3     5       c       vdd     vdd  PMOS  l=2e-6 w=3e-6
MP4     out     d       5       vdd  PMOS  l=2e-6 w=3e-6
MP5     6       e       5       vdd  PMOS  l=2e-6 w=3e-6
MP6     7       f       6       vdd  PMOS  l=2e-6 w=3e-6
MP7     out     g       7       vdd  PMOS  l=2e-6 w=3e-6
MN1     2       e       gnd     gnd  NMOS  l=2e-6 w=3e-6
MN2     2       f       gnd     gnd  NMOS  l=2e-6 w=3e-6
MN3     2       g       gnd     gnd  NMOS  l=2e-6 w=3e-6
MN4     out     d       2       gnd  NMOS  l=2e-6 w=3e-6
MN5     out     a       4       gnd  NMOS  l=2e-6 w=3e-6
MN6     4       b       3       gnd  NMOS  l=2e-6 w=3e-6
MN7     3       c       gnd     gnd  NMOS  l=2e-6 w=3e-6
.ends gate1

.subckt gate2 a b c d e out vdd
MP1     2       d       vdd     vdd  PMOS  l=2e-6 w=3e-6
MP2     2       e       vdd     vdd  PMOS  l=2e-6 w=3e-6
MP3     3       a       2       vdd  PMOS  l=2e-6 w=3e-6
MP4     out     b       3       vdd  PMOS  l=2e-6 w=3e-6
MP5     out     c       2       vdd  PMOS  l=2e-6 w=3e-6
MN1     4       a       gnd     gnd  NMOS  l=2e-6 w=3e-6
MN2     4       b       gnd     gnd  NMOS  l=2e-6 w=3e-6
MN3     5       d       gnd     gnd  NMOS  l=2e-6 w=3e-6
MN4     out     e       5       gnd  NMOS  l=2e-6 w=3e-6
MN5     out     c       4       gnd  NMOS  l=2e-6 w=3e-6
.ends gate2

.subckt inv in out vdd
MP1     out     in      vdd     vdd  PMOS l=2e-6 w=3e-6
MN1     out     in      gnd     gnd  NMOS l=2e-6 w=3e-6
.ends inv

X1 a b c 20 a b c 10            vdd     gate1
X2 a b c a b 20                 vdd     gate2
X3 20 carry                     vdd     inv
X4 10 sum                       vdd     inv

*interface a                    orientation N
*interface b                    orientation N
*interface c                    orientation N
*interface carry                orientation S
*interface sum                  orientation S
.end
```

Figure 3.16 - Description Spice hiérarchisée

Si nous utilisons une bibliothèque structurelle (fichier exemple dans l'annexe 2), la partie de définition de sous-circuits est remplacée par la commande "*.include librairie*".

3.3.3 Description pour la simulation

Il est recommandé d'avoir deux fichiers Spice: le fichier **contrôle** (figure 3.17), qui contient les paramètres de la simulation électrique et le fichier **topologique** (figures 3.15 ou 3.16), qui contient la description du circuit au niveau transistors ou portes. Cette séparation permet la simulation et le dimensionnement du circuit sans édition manuelle des paramètres de la simulation après chaque génération de masques de la macro-cellule. La syntaxe du fichier Spice pour la simulation est définie ci-dessous.

- Temps d'analyse : *.tran tincr tstop*
- Signal continu d'alimentation : *VCC vdd! gnd! dc 5.*
  Les noeuds *vdd!* et *gnd!* sont donnés par l'extraction CADENCE.
- Description des entrées primaires du circuit: *VCx x gnd! PULSE(v1 v2 td tr tf tw tper)*
- Description des sorties primaires du circuit : *.print tran V(x) V(y) ...*
- Inclusion du fichier qui contient les transistors et les capacités parasites (généré par un extracteur électrique, par exemple CADENCE).
  .include extract
- Les "réferences" pour le dimensionnement et la simulation
  MTHn d g s substrat model w l
  Les réferences doivent avoir obligatoirement un nom commençant par MTH.
- La charge sur les noeuds de sortie : *Cx x gnd! 100f*

```
*adder
.tran 0.5n 200n
.options post      * sortie pour l'interface graphique de HSPICE

vcc     vdd!    gnd! dc 5
vea     100     gnd! pulse(5 0 0 .1n .1n 30n 60n)
veb     b       gnd! dc 5
vec     c       gnd! dc 5
.print tran v(sum) v(carry)

.include extract  * fichier spice à plat résultant de l'extraction électrique

MTH1 a 100 gnd! gnd!    Model3 l=2e-6 w=3e-6
MTH2 a 100 vdd! vdd!    Model2 l=2e-6 w=3e-6

C1      sum     gnd!  100fF
C2      carry   gnd!  100fF

.end
```

Figure 3.17 -Description Spice pour la simulation électrique

3.4 ISOLATION DE SOUS-RESEAUX [MOR90]

L'outil accepte soit une description Spice à plat (au niveau transistors), soit une description hiérarchisée (au niveau portes). Dans les deux cas, le générateur doit trouver tous les éléments de base pour réaliser la synthèse du dessin des masques. Ces éléments de base sont :

- les portes de transmissions,
- les cellules élémentaires (représentées par un graphe).

Les portes de transmission sont définies par une connexion parallèle entre un transistor N et un transistor P. Les cellules élémentaires sont les portes *nand*, *nor*, *inverseur*, *AOI* ("*and-or-inverter*") et *OAI* ("*or-and-inverter*"). Elles sont définies par les caractéristiques suivantes :

- logique statique,
- un ensemble de *n* entrées et une sortie,
- même nombre de transistors N et P,
- connexions série ou parallèle entre les transistors,
- plan N dual du plan P.

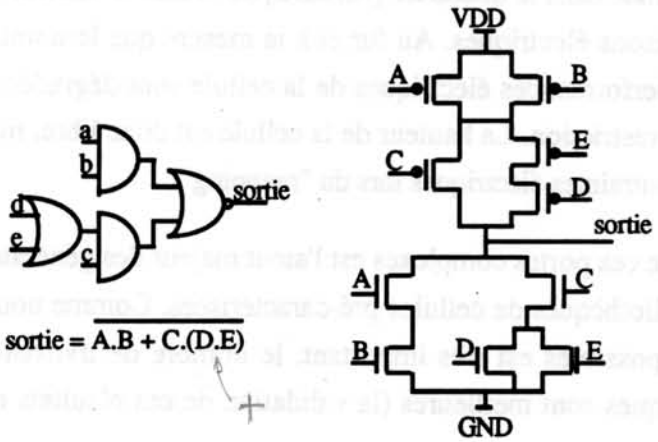


Figure 3.18 - Exemple de cellule élémentaire

La description d'entrée est systématiquement mise à plat, même si celle-ci est déjà hiérarchisée. Ensuite la hiérarchie est reconstruite au niveau portes élémentaires et portes de transmission. Ceci est dû au fait qui il existe des cellules qui sont composées par un ensemble de différentes cellules. Par exemple, une bascule D est composée par deux *inverseurs* et deux portes de transmission, une porte *and* est composée par un *nand* et un *inverseur*.

L'utilisation des portes complexes, *AOI* ou *OAI*, permet de réduire le nombre de transistors et le nombre de couches logiques du circuit lors de l'implantation d'une fonction logique [ABO92]. L'exemple de la figure 3.18 montre un cas typique de circuit numérique.

Pour l'implantation de cette fonction à l'aide de portes *and* ou *or* il est nécessaire d'utiliser au moins 20 transistors, en 3 couches logiques. En utilisant une porte complexe, 10 transistors sont nécessaires, en 1 couche logique. L'utilisation de portes complexes permet donc d'améliorer les performances électriques d'un circuit, tout en ayant une surface occupée moindre [MAZ87].

Le tableau 3.1 [DET87] présente le nombre de portes complexes différentes qui peuvent être générées selon la *hauteur* de la porte complexe. La *hauteur* d'une porte complexe est définie comme le plus grand nombre de transistors en série, soit dans le plan N soit dans le plan P. La cellule de la figure 3.18 a une hauteur de 3, car elle possède trois transistors en série dans le plan P.

Hauteur de la cellule	Nombre de cellules
1	1
2	7
3	87
4	3503
5	425803
6	154793519

Tableau 3.1 - Nombre de portes complexes selon la hauteur de la cellule

Il est recommandé dans la littérature [MAZ92] de limiter la hauteur de portes complexes à quatre, pour des raisons électriques. Au fur et à la mesure que le nombre de transistors en série augmente, les performances électriques de la cellule sont dégradées. Nous n'avons pas tenu compte de cette restriction. La hauteur de la cellule est donc libre, mais le concepteur ne doit pas oublier les contraintes électriques lors du "mapping".

L'utilisation de ces portes complexes est l'atout majeur des générateurs de masques qui n'utilisent pas de bibliothèques de cellules pré-caractérisées. Comme nous l'avons montré, le nombre de cellules possibles est très important, le nombre de transistors est réduit et les performances électriques sont meilleures (la validation de ces résultats est effectuée dans le chapitre 5).

Pour isoler les éléments de base, les étapes principales utilisées par l'algorithme sont :

procédure isolation_de_sous-reseaux

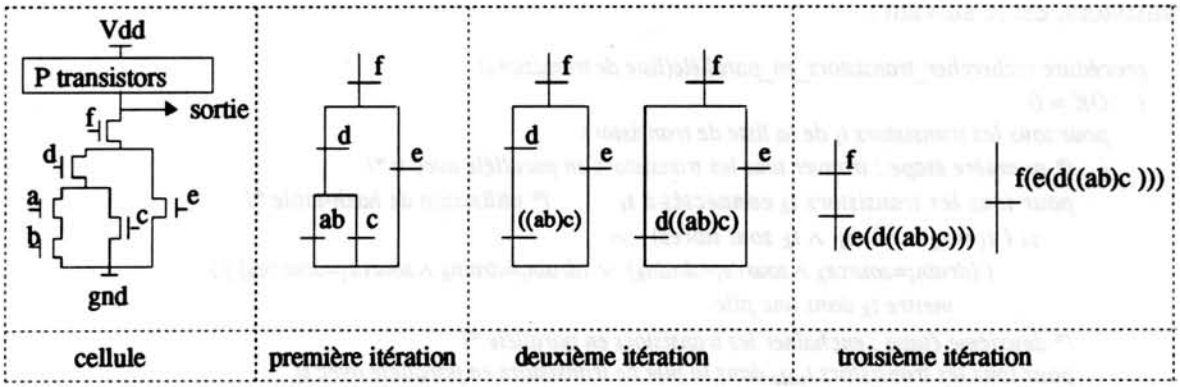
{
- *lecture du fichier SPICE (à plat ou hiérarchisé)*
- *si le fichier SPICE est hiérarchisé, le mettre à plat*
- *déterminer les sorties des cellules*
- *isoler les sous-réseaux N (procédure rechercher_cellules(tous les transistors N))*
- *isoler les sous-réseaux P (procédure rechercher_cellules(tous les transistors P))*
- *isoler les portes de transmission*
- *grouper le sous-réseaux N et P appartenant à la même cellule*

}

Après la lecture du fichier SPICE et son éventuelle mise à plat, nous obtenons deux listes de transistors: la liste N et la liste P. Chaque cellule de base est représentée par deux graphes, le graphe N et le graphe P. Les extrémités de chaque graphe sont l'alimentation et la sortie de la cellule. Nous devons donc construire la liste de toutes les sorties du circuit avant d'isoler les cellules de base. Cette liste des sorties est obtenue en cherchant les noeuds communs entre les transistors N et P. Pour isoler les sous-réseaux, nous allons maintenant chercher de façon itérative, sur chaque liste de transistors, toutes les liaisons série ou parallèle. L'algorithme de base est le suivant:

```
procédure rechercher_cellules( liste de transistors )
{
    tant que ( rechercher_transistors_en_parellèle( liste de transistors ) ∨
               rechercher_transistors_en_serie( liste de transistors )
             );
}
```

La figure 3.19 montre l'exécution de l'algorithme sur les transistors N d'une cellule *AOI*. Le nombre d'itérations est égal au nombre de niveaux logiques dans la cellule. Dans l'exemple, trois itérations sont nécessaires, puisque cette cellule *AOI* contient 3 niveaux logiques. Pour les cellules plus simples (*nand*, *nor* et *inverseur*), une seule itération est nécessaire. L'algorithme d'isolation des sous-réseaux cherche toutes les cellules élémentaires présente dans la liste des transistors a un instant donné. Ainsi, le nombre total d'itérations nécessaires pour exécuter l'algorithme est égal au nombre de niveaux logiques de la cellule la plus complexe.



	1ère itération	2ème itération	3ème itération
transistors en parallèle	-	((ab)c)	(e(d((ab)c)))
transistors en série	ab	d((ab)c)	f(e(d((ab)c)))

Figure 3.19 - Exécution du algorithme d'isolation de sous-réseaux

Les sous-réseaux, déterminés par l'algorithme, sont représentés en utilisant une notation de parenthèses [KWO88]. Cette notation est déterminée à l'aide des règles suivantes:

- Les éléments en parallèle sont mis entre parenthèses. Si un des éléments en parallèle est déjà une association de transistors, cet élément est mis également entre parenthèses. Dans l'exemple, lors de la deuxième itération, il y a une liaison parallèle entre 'ab' et 'c'. Le résultat et donc '((ab)c)', où la première parenthèse indique la liaison parallèle, et la deuxième la liaison série.
- Pour les liaisons série les éléments ne sont pas mis entre parenthèses.

Cette représentation permet d'identifier les portes dans un fichier SPICE, à condition que le fichier ne contienne que des portes en logique statique et des portes de transmission. Par exemple, si pour les transistors N nous obtenons les chaînes: 'a', 'a b c', '(a b)', '(a b) (c d)' et '((a b) (c d))'. Les portes sont :

- Inverseur : seule porte que ne contient qu'un seul transistor entre l'alimentation et la sortie.
- Nand avec 3 entrées : l'absence de parenthèses caractérise la liaison série.
- Nor avec 2 entrées : la parenthèse caractérise la liaison parallèle.
- AOI (a+b)(c+d) : série de deux liaisons parallèles.
- OAI (a.b)+(c.d) : parallèle de deux liaisons série.

Pour que deux transistors 't_i' et 't_k' soient en parallèle, la condition est la suivante:

$$(drain_i = drain_k \wedge source_i = source_k) \vee (drain_i = source_k \wedge source_i = drain_k)$$

L'algorithme qui permet de trouver les transistors en parallèle, à partir d'une liste de transistors, est le suivant :

```
procédure rechercher_transistors_en_parallèle(liste de transistors)
{  OK = 0
   pour tous les transistors ti de la liste de transistors
   {  /* première étape : trouver tous les transistors en parallèle avec ti */
      pour tous les transistors tk connectés à ti        /* utilisation de hash-table */
         si ( ti ≠ tk  ∧  (ti ∧ tk sont libres)  ∧
              ( (draini=sourcek ∧ sourcei=draink) ∨ (draini=draink ∧ sourcei=sourcek) ) )
                 mettre tk dans une pile
      /* deuxième étape : enchaîner les transistors en parallèle */
      pour tous les transistors tpile dans la pile de transistors en parallèle avec ti
      {  • si tpile est déjà une association de transistors, mettre tpile entre parenthèses
         • enchaîner le transistor tpile à ti
         • signaler le transistor tpile comme occupé /* c'est à dire, non libre */
         • OK = 1
      }
      mettre les éléments en parallèle entre parenthèses
   }
   retourner ( OK )
}
```

Pour que deux transistors soient en série, ils ne doivent être connectés que par un noeud. Ce noeud peut être la connexion entre le drain d'un transistor et la source de l'autre (et réciproquement). Ainsi, les quatre conditions de liaison série sont :

$$drain_i = source_k \vee drain_i = drain_k \vee source_i = source_k \vee source_i = drain_k$$

Ce noeud ne peut être une extrémité du graphe (alimentation ou sortie de la cellule). L'algorithme est le suivant :

```
procédure rechercher_transistors_en_série(liste de transistors)
{  OK = 0
   pour tous les transistors ti de la liste de transistors
   faire
   {  flag=0
      /* première étape : trouver un transistor en série avec ti */
      pour tous les transistors tk connectés à ti        /* utilisation de hash-table */
         si ( ti ≠ tk  ∧  (ti ∧ tk sont libres)  ∧  (ti ∧  tk  ne sont pas en parallèle) )
         {   si ( (drain_i=source_k ∨ drain_i=drain_k)  ∧ drain_i≠alimentation ∧ drain_i≠sortie)
             {   connexions par drain = connexions par drain + 1
                 candidat = k
             }
             si ( (source_i=source_k ∨ source_i=drain_k) ∧ source_i≠alimentation ∧ source_i≠sortie)
             {   connexions par source = connexions par source + 1
                 candidat = k
             }
         }
      /*deuxième étape: vérifier si il y a SEULEMENT une connexion au drain ou à la source de ti */
      si ( le nombre de connexions par drain = 1 )
      {  enchaîner les transistors ti et t_candidat
         signaler le transistor t_candidat comme occupé
         le drain de ti deviendrait la source ou le drain de t_candidat , selon le type de connexion
         flag = OK = 1
      }
      sinon si ( le nombre de connexions par source = 1 )
      {  enchaîner les transistors ti et t_candidat
         signaler le transistor t_candidat comme occupé
         la source de ti deviendrait la source ou le drain de t_candidat , selon le type de connexion
         flag = OK =1
      }
   }
   tant que ( flag = 1 )
   retourner ( OK )
}
```

Pour trouver les portes de transmission, l'algorithme utilisé est analogue à celui permettant de trouver les transistors en parallèle. La différence étant dans le mode de recherche: au lieu de chercher deux transistors de même type en parallèle, l'algorithme cherche deux transistors de type différent en parallèle.

Le temps d'exécution nécessaire à la recherche de transistors séries/parallèles ou des portes de transmission est présenté dans le tableau 3.2. L'utilisation des structures du type "*hash-table*", qui permettent de trouver les groupes de transistors connectés, réduit sensiblement le temps de calcul nécessaire pour rechercher les sous-réseaux séries/parallèles.

Nombre de Transistors	Temps d'exécution - secondes (Sun Sparc 10)
540	0,74
1014	1,29
2816	3,15
4476	6,10
10112	16,75
14376	19,15
38342	113,10
43888	141,50

Tableau 3.2 - Temps d'exécution pour isoler les sous-réseaux d'un circuit

La dernière partie de l'algorithme va associer chaque sous-réseau N avec le sous-réseau P correspondant. Les deux sous-réseaux sont groupés si leur noeud de sortie est identique.

Remarque : s'il y a plus de un sous-réseau N ou P avec la même sortie, cela peut signifier un court-circuit. Dans ce cas, le concepteur doit revoir la description d'entrée pour corriger l'éventuel erreur.

Cette procédure d'isolation des sous-réseaux peut être utilisée comme un outil indépendant permettant l'extraction logique. Tout concepteur sait que le résultat d'une extraction électrique est une liste à plat de transistors, qui est difficile à interpréter. Si nous utilisons la procédure d'isolation de sous-réseaux sur cette liste nous obtenons les cellules de base composant le circuit. C'est à dire, qu'à partir de la description au niveau masques, nous pouvons obtenir la description au niveau logique, en utilisant un extracteur électrique et notre "extracteur logique". La généralisation de cette procédure pour reconnaître d'autres topologies (par exemple, logique dynamique), est possible par la définition d'une bibliothèque de règles, permettant la reconstitution de la description structurelle du circuit.

3.5 GENERATION DE CELLULES

Cette étape permet de générer chaque cellule de base de la macro-cellule, dans le style linear-matrix. Les objectifs à atteindre sont:

- largeur optimale,
- hauteur minimale.

La largeur optimale d'une cellule linear-matrix correspond à une implantation sans cassures de diffusion [UEH81]. Pour avoir la hauteur minimale nous devons générer toutes les solutions optimales en largeur, et parmi ces solutions trouver celle qui minimise le nombre de pistes de routage. Selon [MAZ87], ce problème est NP-complet.

Etant donné la faible complexité d'une cellule: entre 2 transistors (inverseur) et 32 (dans la pratique), l'implantation d'un algorithme exact ne pose pas de problème. L'algorithme développé est constitué de 3 étapes:

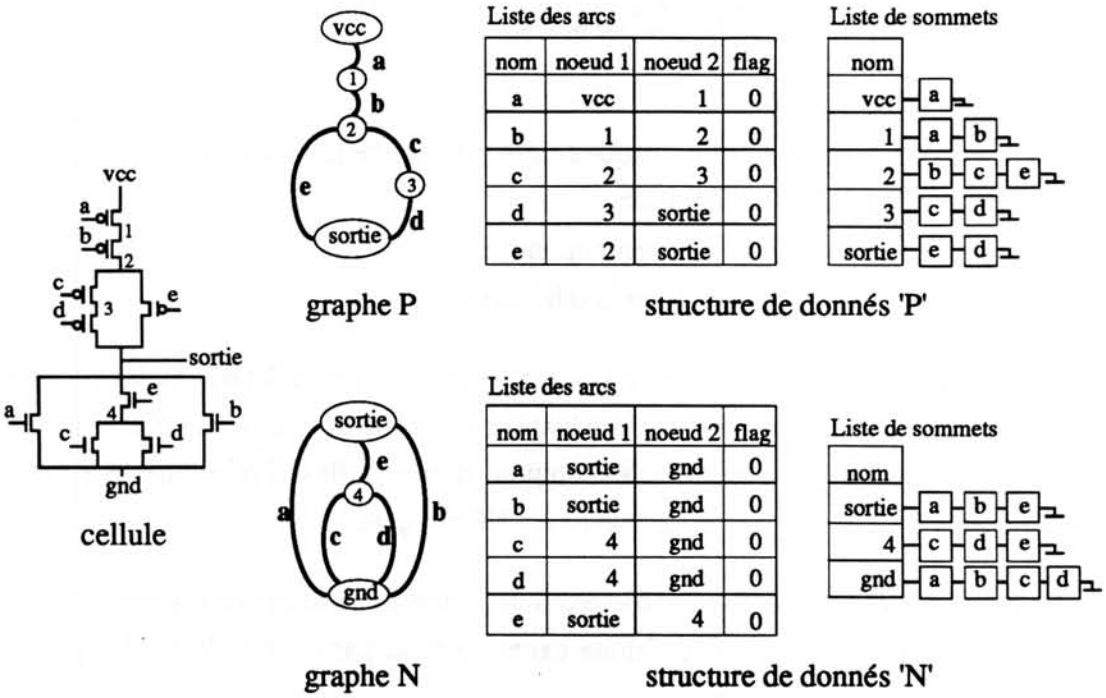
- construction du graphe représentant la cellule,
- construction des solutions candidates (optimales en largeur),
- sélection d'une solution (minimale en hauteur).

a/ Construction du graphe représentant la cellule

Chaque cellule est représentée par deux graphes non orientés, un pour le plan 'N' et un pour le plan 'P'. Les arcs 'a_i' sont les grilles des transistors et les sommets 's_k' les signaux de drain et de source. Les extrémités des graphes sont : l'alimentation ('Vdd' pour le plan 'P' et 'Gnd' pour le plan 'N') et la sortie de la cellule. La figure 3.20 illustre les graphes d'une cellule OAI et la structure de donnés utilisée.

Les graphes sont représentés par deux listes : liste des arcs et liste des sommets. La liste des arcs contient le nom de l'arc ('a_i'), les deux signaux connectés à l'arc ('s_k' et 's_m') et un drapeau qui signalise quand l'arc est déjà visité. La liste de sommets contient les références ("*pointeurs*") aux arcs connectés à chaque sommet.

Cette redondance d'information (deux listes) permet une recherche d'informations plus rapide. Par exemple, étant donné un arc 'a_i' et un de ses sommets 's_k'. Quelles sont les arcs connectés à son deuxième sommet 's_m' (*sommet voisin*) ? Pour savoir qui est 's_m' il suffit de regarder les deux noeuds connectés à 'a_i' dans la liste des arcs. Le sommet 's_m' est le noeud différent de 's_k'. Nous regardons ensuite dans la liste de sommets quels sont les arcs connectés à 's_m'.



nom	noeud 1	noeud 2	flag
a	vcc	1	0
b	1	2	0
c	2	3	0
d	3	sortie	0
e	2	sortie	0

nom	noeud 1	noeud 2	flag
a	sortie	gnd	0
b	sortie	gnd	0
c	4	gnd	0
d	4	gnd	0
e	sortie	4	0

Figure 3.20 - Graphes et structures de donnés représentant une cellule

b/ Construction des solutions candidates

Une solution candidate est une séquence d'arcs contenant tous les arcs du graphe. Cette séquence est appelée chemin d'Euler. S'il existe au moins un chemin d'Euler pour un graphe, les transistors de ce graphe peuvent être implantés dans une ligne de diffusion, sans cassure.

Le point de départ de l'algorithme pour parcourir un graphe non orienté, est un arc 'a_i' quelconque et un somment 's_k' connecté à 'a_i'.

Les deux premières actions sont respectivement : mettre 'a_i' dans un vecteur temporaire et signaler 'a_i' comme déjà visité. Nous vérifions ensuite si tous les arcs ont été déjà visités. Dans ce cas, nous avons trouvé un chemin d'Euler pour le graphe, et nous sauvegardons la solution du vecteur temporaire avec les autres solutions possibles. S'il reste des arcs connectés au *sommet voisin* de 's_k' (le sommet voisin est l'autre sommet connectée au arc 'a_i'), n'ayant été visités, ils doivent être visités. Cette procédure récursive permet de visiter le graphe d'une façon exhaustive, et par conséquence trouver tous les chemins d'Euler existants. Nous libérons finalement 'a_i' et nous le retirons du vecteur temporaire.

L'algorithme (récursif) est le suivant :

procédure visiter_graphe(a_i, s_k)
{ • mettre a_i dans un vecteur temporaire
• signaler a_i comme visité (a_i.flag=1)
• si tous les arcs sont visités
alors : sauvegarder le vecteur temporaire
sinon : pour tous les arcs a_s connectés au sommet voisin de s_k :
visiter_graphe(a_s, s_{voisin})
• retirer a_i du vecteur temporaire
• libérer a_i (a_i.flag=0)
}

Pour illustrer l'algorithme, la figure 3.21 montre l'exécution de l'algorithme pour la cellule exemple (figure 3.20), à partir de l'arc 'a' et du sommet 'sortie'.

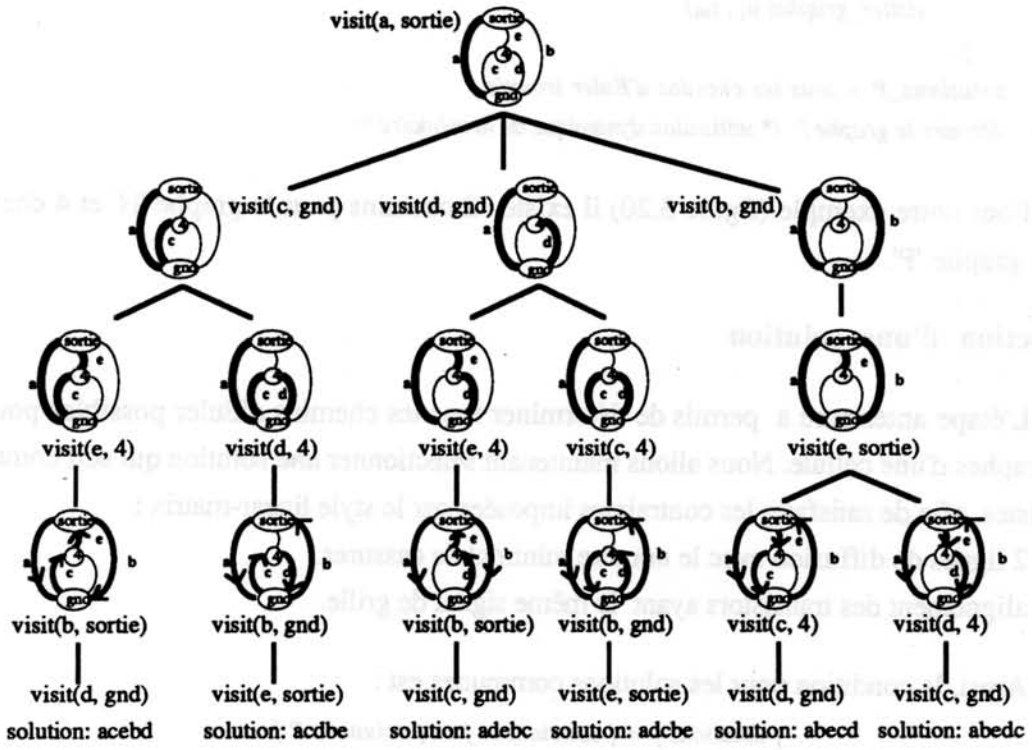


Figure 3.21 : Exécution de l'algorithme de recherche de chemin d'Euler pour la cellule (AB(E(CD))), plan 'N', à partir de l'arc 'a' et du sommet 'sortie'

Nous pouvons remarquer qu'il existe 6 séquences possibles pour ce point de départ (arc 'A' et sommet 'sortie') :

ACEBD ACDBE ADEBC ADCBE ABECD ABEDC

La procédure "*visiter_graphe*" est exécutée pour chaque arc du graphe, à partir des deux sommets connectés à l'arc. Ainsi, la procédure pour construire l'ensemble de solutions optimales en largeur est la suivante:

procédure construire_solutions(cellule)
{
- *construire le graphe N*
- *pour tous les arcs a_i du graphe N*
  { visiter_graphe(a_i , s_k)
  visiter_graphe(a_i , s_m)
  }
- ***solutions_N** = tous les chemins d'Euler trouvés*
- *détruire le graphe N /* utilisation dynamique de la mémoire */*
- *construire le graphe P*
- *pour tous les arcs a_i du graphe P*
  { visiter_graphe(a_i , s_k)
  visiter_graphe(a_i , s_m)
  }
- ***solutions_P** = tous les chemins d'Euler trouvés*
- *détruire le graphe P /* utilisation dynamique de la mémoire */*

}

Pour notre exemple (figure 3.20) il existe 32 chemins pour le graphe 'N' et 4 chemins pour le graphe 'P'.

c/ Sélection d'une solution

L'étape antérieure a permis de déterminer tous les chemins d'Euler possibles pour les deux graphes d'une cellule. Nous allons maintenant sélectionner *une* solution qui soit commune aux 2 listes, afin de satisfaire les contraintes imposées par le style linear-matrix :

- 2 lignes de diffusion avec le nombre minimal de cassures,
- alignement des transistors ayant le même signal de grille.

Ainsi, la condition pour les solutions communes est :

$$\{ \text{solutions} \} = \{ \text{solutions N} \} \cap \{ \text{solutions P} \}$$

S'il existe une ou plusieurs solutions, cela signifie qu'il existe une ou plusieurs solutions sans cassure de diffusion, et par conséquence la cellule correspondante a la largeur optimale. L'objectif à atteindre pour sélectionner une solution parmi cet ensemble, est dans ce cas la minimisation de la hauteur de la cellule. Selon [MAZ92] 56% des cellules avec une hauteur 4 (3503 cellules possibles) ont au moins 1 chemin d'Euler commun pour les deux plans. Dans l'exemple de la figure 3.20, il existe 2 solutions communes : CDEBA et ABEDC.

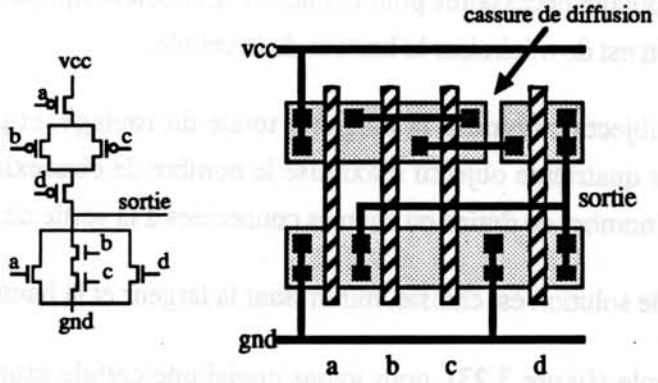
S'il n'y a pas des solutions communes entre les deux graphes, nous prenons toutes les solutions 'N' et 'P' trouvées :

Si ({ solutions N } ∩ { solutions P } = ∅)

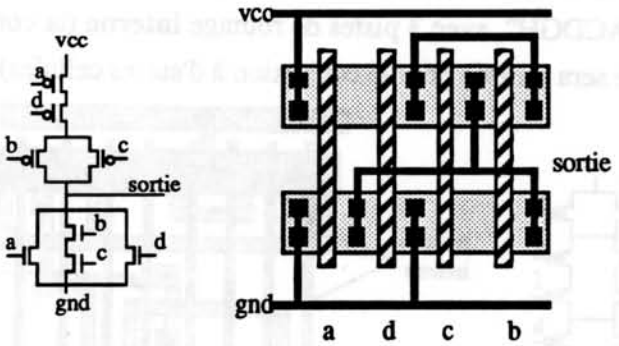
alors : { solutions } = { solutions N } ∪ { solutions P }

Le résultat est le chemin d'Euler dans un plan, et une ou plusieurs cassures de diffusion dans l'autre plan. La première fonction objective est donc la minimisation de cassures de diffusion, et la seconde la minimisation de la hauteur de la cellule.

Le dernier cas (absence de solutions 'N' et 'P') l'ordre de transistors est celui obtenu lors de l'extraction de sous-réseaux, qui a déjà groupé les liaisons série et parallèle. Le changement d'ordre des transistors ("*reordering*") peut permettre la génération de la cellule sans cassures de diffusion. La figure 22 montre un exemple où le "reordering" peut être utile. Le circuit original n'a pas de chemin d'Euler dans le plan 'P'. Si le concepteur change l'ordre des transistors dans ce plan, les deux plans peuvent être générés sans cassure de diffusion.



(a) cellule sans chemin d'Euler dans le plan 'P'



(b) changement de l'ordre de transistors pour avoir une solution sans cassures de diffusion

Figure 3.22 - Technique de "*reordering*" pour trouver le chemin d'Euler dans une cellule

Plusieurs articles [LEF89][MAD89][CC88] proposent des algorithmes permettant d'exécuter le "*reordering*" de façon automatique. Nous n'avons pas implanté le "*reordering*" dans notre logiciel, car le but de notre générateur est la génération d'une macro-cellule, constituée des centaines de cellules. Dans l'ensemble, l'effet de quelques cellules non optimales est négligeable.

Après avoir sélectionné une ou plusieurs solutions candidates, nous passons à la dernière étape : sélection d'une solution selon un ensemble de fonctions objectifs:

a/ minimiser le nombre de cassures de diffusion (largeur)

b/ minimiser la densité de connexions (hauteur)

c/ minimiser les longueurs de connexions (capacités parasites)

d/ maximiser le nombre de liaisons à l'alimentation.

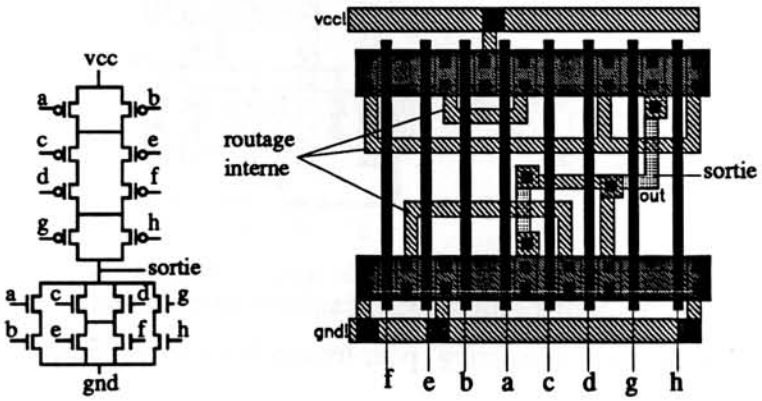
Le premier objectif, minimiser le nombre de cassures de diffusion, permet de réduire la largeur de la cellule et les capacités parasites (surtout celle de périphérie, appelé "*locox*"). Robert, dans [ROB91], a montré qu'une cellule sans cassure sur les lignes de diffusion a le minimum de capacités parasites.

Le deuxième objectif, minimiser la densité de connexions, permet de diminuer le nombre de pistes de routage nécessaires pour connecter les noeuds équipotentiels de la cellule. Le but de cette fonction est de minimiser la hauteur de la cellule.

Le troisième objectif minimise la longueur totale du routage, et par conséquence les éléments parasites. Le quatrième objectif maximise le nombre de connexions à l'alimentation, ceci afin de réduire le nombre de drains ou sources connectées à la sortie de la cellule.

Ainsi, une seule solution est choisie, minimisant la largeur et la hauteur de la cellule.

Comme exemple (figure 3.23), nous avons choisi une cellule proposée par [MAZ92], où l'algorithme génère une solution optimale en largeur et hauteur. Notre algorithme a trouvé la même solution, "FEBACDGH", avec 3 pistes de routage **interne** (la connexion de sortie n'est pas considérée, car elle sera utilisée pour la connexion à d'autres cellules).



total solutions N : 256 - total solutions P: 64 - total solutions possibles (N ∩ P): 16

Figure 3.23 -Exemple de cellule généré automatiquement

Le résultat de l'étape de génération de cellules est l'ordre des transistors pour chaque cellule de la macro-cellule. Sous l'étape suivante de placement il est nécessaire de partitionner le circuit en bandes, puis placer les cellules pour enfin les connecter (routage).

3.6 PLACEMENT

Le problème de positionnement de cellules (placement) consiste en une transposition d'une description structurelle en une description physique (position des cellules dans le plan de masques du circuit). Cette description physique comprend : une paire de coordonnées et l'orientation de la cellule (rotation et pivotement). La qualité du placement conditionne celle du routage : un mauvais placement peut conduire à un routage avec des connexions très longues, une surface occupée excessive et des performances très dégradées.

Parmi les grandes classes d'algorithmes de placement [PRE88], citons :

- Les algorithmes **constructifs** [SCH72] [SCH76] [AKE81] [KAN83], qui partent des contraintes utilisateur ou d'un placement partiel avec des cellules "graines", en plaçant successivement les différents éléments selon un processus agrégatif ascendant (dit "*bottom-up cluster grow*"), en fonction de l'affinité aux éléments déjà placés.
- Les algorithmes **itératifs**, qui à partir d'un placement initial, engendrent d'autres placements par des transformations simples (par exemple, échange des positions) qui sont évalués selon certains critères, puis acceptés ou refusés de façon déterministe ou aléatoire, avec possible retour-arrière ("*back-tracking*"). De très nombreuses variantes en ont été élaborées, parmi lesquelles les méthodes de relaxation [PIL88], de recuit-simulé [KIR83] et les algorithmes génétiques [ESB92]. Elles sont utilisées tant pour créer un placement initial que pour optimiser ou rendre valide un placement issu d'une autre méthode.
- Les algorithmes qui affinent progressivement la localisation des cellules. Ils procèdent usuellement de façon descendante ("*top-down*") par **partitionnement** récursif de la surface du circuit (bissections ou quadrissections), en cherchant à chaque étape à minimiser le nombre de connexions reliant les zones partitionnées ([KER70] [BRE77] [PAT81] [FID82] [KAM82] [KOZ83] [CHA90]).

Pour résoudre le problème de positionnement dans notre générateur nous avons adopté une démarche qui combine ces trois catégories d'algorithmes. Dans notre approche, la macro-cellule étant supposée découpée en un certain nombre de **bandes** (soit spécifié par l'utilisateur, soit calculée), nous procédons succinctement selon les étapes suivantes, qui seront détaillées dans la suite :

- Par un processus agrégatif (constructif), nous constituons des groupes contenant des cellules fortement connexes (item 3.6.1.a).
- Ces groupes sont alors assignés aux bandes, avec pour objectifs de minimiser les connexions (verticales) entre les bandes et d'homogénéiser l'occupation des bandes (item 3.6.1.b).

- Puis, par un processus itératif, nous équilibrons les longueurs de bandes (item 3.6.1.c).
- Enfin, nous procédons au positionnement fin des cellules dans chaque bande conjointement au routage des connexions internes à la bande, tout en prenant en compte les connexions inter-bandes vers les bandes déjà traitées. Pour cela, les bandes sont successivement traitées selon un ordre qui sera explicitée (item 3.6.2).

3.6.1 Partition en bandes

La partition du circuit [DUN 85] est faite selon trois fonctions objectif :

- nombre minimal de connexions entre les bandes: condition nécessaire pour minimiser la longueur du routage et par conséquence optimiser les performances électriques du circuit. La littérature donne une très grande importance à cette condition, car elle aboutit à une génération orientée par les performances [PED91].
- même largeur pour chaque bande : condition nécessaire pour avoir une utilisation optimale de la surface.
- respect de l'orientation des broches d'entrée/sortie du circuit.

L'algorithme implanté comprend trois étapes :

- formation de groupes,
- positionnement des groupes,
- équilibrage des longueurs de bandes.

Les deux premières étapes de l'algorithme sont constructives. La troisième étape est un algorithme itératif, qui va améliorer la solution obtenue par les deux premières étapes.

a/ Formation de groupes

Le but de cette étape, est de grouper les cellules le plus fortement connectées.

Nous avons défini le paramètre "coût externe" comme le nombre de connexions externes à un groupe de cellules (ou groupes), qui sont connectées par une liaison commune. La liaison commune à un groupe de cellules (ou groupes) doit connecter au maximum **2 ou 3** cellules (ou groupes). Ceci est fait afin d'éviter la formation de groupes dominants [SCH72]. Cette fonction 'f' est évaluée pour toutes les liaisons 'net_k' du circuit.

$$f(net_k) = | \{n_a\} \cup \{n_b\} \cup \{n_c\} - \{net_k\} |$$

où: $\{n_i\}$ - **ensemble** des connexions de chaque cellule

$\{net_k\}$ - connexion commune aux cellules '*a*', '*b*' et '*c*'

Les cellules (ou groupes) connectées à la liaison ayant le plus petit coût externe sont groupées. La figure 3.24 montre deux cas distincts. Dans le premier, le coût externe de connecter les cellules 'a' et 'b', par la liaison '3' (net_k) est 3. C'est à dire, que si nous groupons les cellules 'a' et 'b' il y aura 3 connexions **externes** à ce groupe. Dans le deuxième, le coût du groupement des cellules 'a', 'b' et 'c' (net_k = '6') est égal à 4.

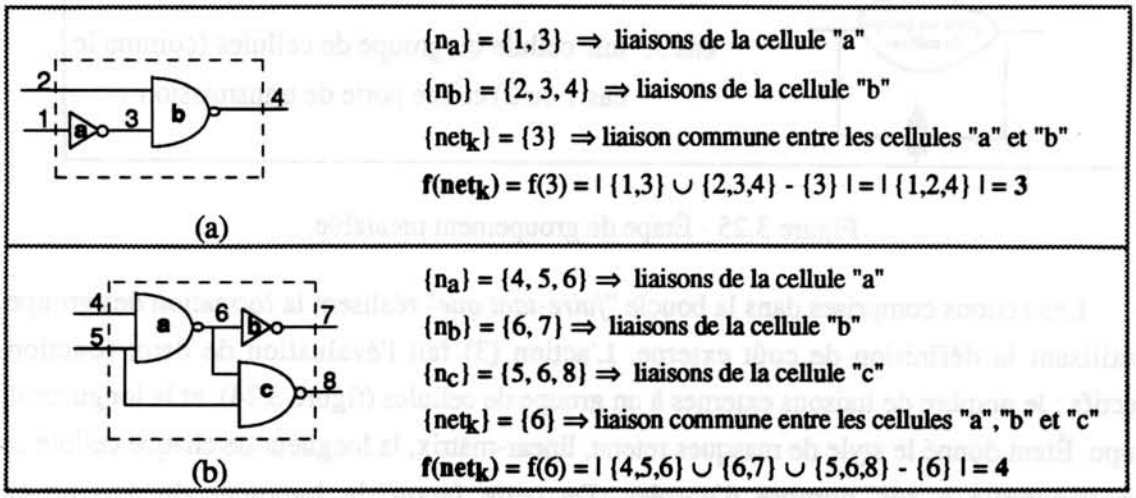


Figure 3.24 - Groupement de cellules selon la fonction coût externe

Les principales étapes de l'algorithme sont :

```
procédure cluster ( liste de cellules )
{  pre_cluster (liste de cellules)
   faire
   {  (1) OK = 0
      (2) construire la liste de liaisons entre les cellules
      (3) pour chaque liaison net_k dans la liste de liaisons, qui connecte 2 ou 3 cellules (ou groupes)
            {  évaluer la fonction coût externe : coût
               évaluer la longueur du groupe : longueur
            }
      (4) pour la liaison net_k avec le coût minimum ET la longueur minimale :
            OK = grouper les cellules connectées à net_k
      (5) détruire la liste de liaisons
   }
   tant que ( OK ≠ 0 )
}
```

L'étape "*pre_cluster*" (groupement préalable), réduit le nombre d'éléments à traiter, en réalisant le groupement par **paires**, des cellules qui ont une liaison exclusive. Le cas typique dans les circuits numériques (figure 3.25, cas 1 et 2) se produit entre une porte 'inverseur' et une porte quelconque. Le cas 3 est un cas particulier, mais que l'on peut rencontrer souvent (utilisation de bascules).

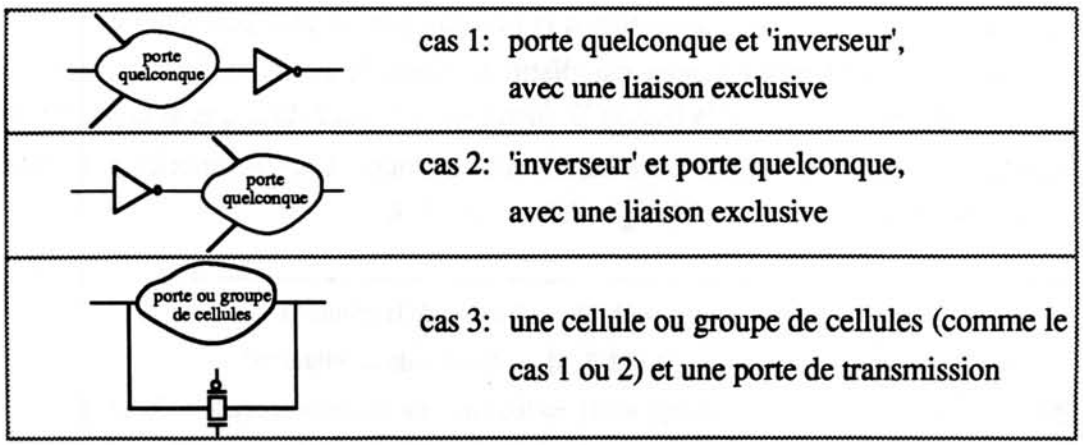


Figure 3.25 - Étape de groupement préalable

Les actions comprises dans la boucle "*faire-tant que*" réalisent la formation des groupes en utilisant la définition de coût externe. L'action (3) fait l'évaluation de deux fonctions objectifs : le nombre de liaisons externes à un groupe de cellules (figure 3.24), et la longueur du groupe. Étant donné le style de masques retenu, linear-matrix, la longueur de chaque cellule est proportionnelle à son nombre d'entrées. De cette façon, la longueur du groupe est proportionnelle à la somme du nombre d'entrées des cellules du groupe.

Dans l'action (4) nous vérifions s'il est possible de grouper les cellules connectées à la liaison ayant le coût externe et la longueur minimale. Si la longueur du groupe dépasse une valeur pré-déterminée (par exemple 60% de la longueur calcule pour une bande), le groupement n'est pas réalisé. Ainsi, s'il n'y a pas de groupement ou des liaisons qui connectent 2 ou 3 cellules (ou groupes), le drapeau "OK" vaut zéro, et ainsi le processus de groupement est arrêté.

Définition: *la longueur de chaque bande est calculée selon la fonction suivante :*

$$\textit{Largeur de la bande} = \frac{\textit{Nombre de transistors } N}{\textit{Nombre de bandes}}$$

La liste de liaisons est ensuite détruite (action (5)) pour après être de nouveau refaite (action (2)). Ces actions sont nécessaires, car au moment du groupement de cellules les liaisons **internes** au groupe disparaissent, et une nouvelle cellule résultant du groupement est formée.

b/ Assignation des groupes aux bandes

Après la formation des groupes, chaque groupe est placé dans une bande spécifique.

L'algorithme implanté construit un ensemble de solutions candidates pour chaque bande. La sélection de solutions candidates est analogue à celle utilisée dans l'algorithme de

positionnement de cellules (item 3.6.2). Pour chaque solution candidate, l'algorithme évalue les 3 fonctions objectifs, en choisissant la solution qui:

(1) minimise le nombre de connexions inter-bandes (externes aux bandes),

(2) minimise la différence entre la longueur de la solution candidate et la longueur calculée,

(3) place les groupes de cellules ayant des broches avec des contraintes de positionnement sud (ou nord) dans les bordes inférieures (ou supérieures) de la macro-cellule.

c/ Homogénéisation des longueurs des bandes

Cette dernière étape va déplacer les cellules entre différentes bandes, afin de minimiser la surface occupée, en homogénéisant la longueur des bandes (figure 3.26).

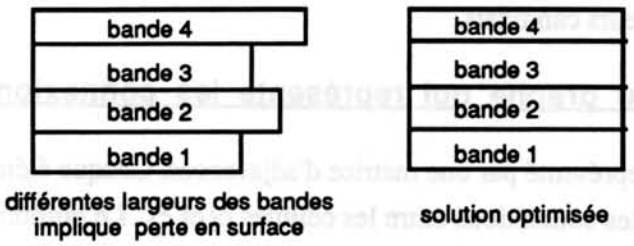


Figure 3.26- Homogénéisation des longueurs des bandes

La cellule choisie pour être déplacée d'une bande plus large vers une autre bande plus courte, sera celle ayant le plus haut gain :

$$gain(c_{ij}) = coût\ externe\ (c_{ij}) - coût\ interne\ (c_i)$$

avec :
- *coût externe* (c_{ij}) : nombre de liaisons de la cellule 'c' externes à la bande 'i' et présentes dans la bande 'j'
- *coût interne* (c_i) : nombre de liaisons de la cellule 'c' exclusives à la bande 'i'

Les déplacements entre deux bandes (adjacentes ou non) sont faites jusqu'à ce qu'il n'y ait plus de déplacements possibles, ou lorsque la longueur calculée est atteinte (dans la pratique il est toléré une marge de 5%, car les cellules ont des longueurs différentes).

Si l'utilisateur ne définit pas le nombre de bandes, l'algorithme tente d'obtenir une macro-cellule de format carré. Dans ce cas, le nombre de bandes est calculé selon l'équation :

$$Nombre\ de\ bandes = \sqrt{\frac{Nombre\ de\ transistors\ N}{Mt}}$$

Définition: *'Mt' est la hauteur moyenne (symbolique) d'une bande, et comprendre la somme des largeurs des transistors et le nombre moyen de pistes de routage (ce nombre a été fixé à 10, à partir d'études statistiques [REI88]).*

Cette procédure de partition a pour résultat une liste de cellules par bande. Cette liste est ensuite utilisée dans les étapes de positionnement et de routage des cellules.

3.6.2 Positionnement des cellules dans chaque bande

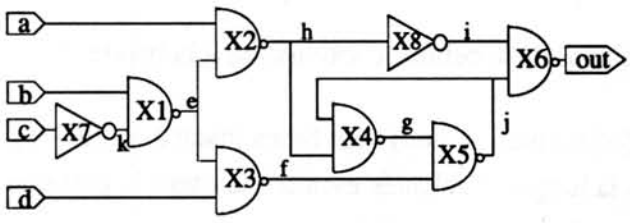
Pour résoudre le problème du placement des cellules dans chaque bande, nous avons implanté un algorithme constructif, qui utilise au départ un ensemble de cellules *graines* afin de générer le placement final. L'algorithme implanté comprend quatre étapes :

- construction du graphe qui représente la connexion entre les cellules,
- choix de *graines*,
- construction d'**arbres de liaisons** (structure de données que représente les connexions entre les cellules d'une même bande),
- évaluation de vecteurs candidats.

a/ Construction du graphe qui représente les connexions entre les cellules

Le graphe est représenté par une matrice d'adjacences. Chaque élément e_{ij} de la matrice représente le nombre des connexions entre les cellules c_i et c_j. Le nombre total de connexions d'une cellule c_k est obtenu par l'addition des éléments de la ligne $l_{k.}$.

La figure 3.27 montre un circuit exemple et la matrice d'adjacences qui représente les connexions de ce circuit. Dans l'exemple, la cellule 'X1' est connectée aux cellules 'X2','X3,'X7'; la cellule 'X2' est connectée aux cellules 'X1','X3','X4','X8' et ainsi de suite.



circuit

cellule	X1	X2	X3	X4	X5	X6	X7	X8
X1	*	1	1	0	0	0	1	0
X2	1	*	1	1	0	0	0	1
X3	1	1	*	0	1	0	0	0
X4	0	1	0	*	2	1	0	1
X5	0	0	1	2	*	1	0	0
X6	0	0	0	1	1	*	0	1
X7	1	0	0	0	0	0	*	0
X8	0	1	0	1	0	1	0	*
total:	3	4	3	5	4	3	1	3

matrice d'adjacences

Figure 3.27 - Circuit et son graphe de connexions

Généralement, les signaux de grilles des transistors 'N' et 'P' dans les portes de transmission sont communs à toutes les portes de transmission du circuit, ils correspondent par exemple, aux signaux d'horloge. Pour éviter qu'ils perturbent inutilement le graphe de connexions, ils sont ignorés au moment de la construction du graphe.

b/ Choix de *graines*

Les cellules graines sont les cellules utilisées comme point de départ au moment de la construction de la structure qui représente les liaisons entre les éléments composant le circuit (**arbre des liaisons**). Nous construisons un vecteur à partir des cellules *les moins connectées* (ces cellules sont celles qui ont une plus grande probabilité de générer une solution avec toutes les cellules de la bande). La figure 3.28 montre ce vecteur pour le circuit exemple. La cellule 'X7' contient 1 liaison, les cellules 'X1','X3','X6','X8' 3 liaisons, 'X2','X5' 4 liaisons et 'X4' 5 liaisons.

nombre de liaisons	1	3				4		5
cellules	X7	X1	X3	X6	X8	X2	X5	X4

Figure 3.28 - Vecteur avec les cellules *graine*

c/ Construction des arbres de liaisons

Le but de cette procédure est de chercher les vecteurs qui maximisent les aboutements entre les cellules du circuit. A partir du graphe de connexions et du vecteur graine, nous construisons de façon récursive un ensemble de solutions candidates au placement. La procédure (récursive) est la suivante :

```
procédure arbre_de_liaisons (vecteur_graine)
{  pour chaque cellule c_graine dans le vecteur_graine:
   {  (1)   insérer la cellule c_graine dans le vecteur_candidat;
      (2)   créer un nouveau vecteur graine avec les cellules connectées à c_graine
            (index de la matrice d'adjacences différent de zéro) et qui ne sont pas dans le vecteur_solution;
      (3)   arbre_de_liaisons( nouveau vecteur) ;          /* appel récursive */
      (4)   score = nombre de cellules dans le vecteur_candidat;
            si ( score < meilleur_score )
            {  meilleur_score = score ;
               effacer toutes les solutions antérieures ;
               garder la solution actuelle ;
            }
            si ( score = meilleur score )
               garder la solution actuelle ;
      (5)   enlever c_graine du vecteur_candidat;
   }
}
```

La ligne '3' réalise la construction de l'arbre de connexions entre les cellules, par l'appel récursif de la fonction "arbre de liaisons". La ligne '4' de la procédure évalue le vecteur candidat au placement, où la fonction objectif est le nombre de cellules dans ce vecteur. Toutes les solutions qui contiennent le plus grand nombre de cellules sont sauvegardées. La figure 3.29 montre les premiers pas de la construction de l'arbre de liaisons.

cellule graine	*vecteur candidat*	*nouveau vecteur candidat*	*arbre de liaisons*
7	7	1 cellule '1' est connectée à '7'	
1	7,1	2,3	
	'2' et '3' sont les cellules connectées à '7' qui ne sont pas dans le vecteur solution.		
2	7,1,2	3,4,8	
3	7,1,2,3	5	
5	7,1,2,3,5	4,6	
4	7,1,2,3,5,4	6,8	
6	7,1,2,3,5,4,6	8	
8	7,1,2,3,5,4,6,8 stocker cette solution, plus de cellules connectées	- enlever la cellule '8' du vecteur solution	
6	7,1,2,3,5,4,6	- enlever la cellule '6' du vecteur solution	
8	7,1,2,3,5,4,8 '8' est l'autre cellule connecté à '4'	6	
6	7,1,2,3,5,4,8,6 stocker cette solution, plus de cellules connectées	- enlever la cellule '6' du vecteur solution	
8	7,1,2,3,5,4,8	- enlever la cellule '8' du vecteur solution	
4	7,1,2,3,5,4	- enlever la cellule '4' du vecteur solution	
6	7,1,2,3,5,6 '6' est l'autre cellule connectée à '5'	4,8	

Figure 3.29 - Premiers pas de l'exécution de l'algorithme de construction de l'arbre des liaisons

Les premiers pas de l'exécution de l'algorithme de construction de l'arbre des liaisons, présentés dans la figure 3.29, ont généré les solutions '7 1 2 3 4 6 8' (huitième itération) et '7 1 2 3 4 8 6' (onzième itération). Nous présentons ci-dessous quelques solutions possibles à partir des cellules du vecteur des "graines" :

(7 1 3 5 6 8 4 2)	(7 1 2 3 5 4 6 8)	(8 6 4 5 3 2 1 7)
(8 4 6 5 3 2 1 7)	(6 8 4 5 3 2 1 7)	(6 5 4 8 2 3 1 7)
(3 5 6 8 4 2 1 7)	(3 5 6 4 8 2 1 7)	(5 6 8 4 2 3 1 7)
(5 4 6 8 2 3 1 7)	(2 4 8 6 5 3 1 7)	(4 2 8 6 5 3 1 7) ...

Cette technique de construction de l'arbre de liaisons permet d'obtenir un nombre **réduit** de solutions possibles au placement des cellules par rapport au nombre de combinaisons possibles entre les cellules (n!). Dans notre exemple, la construction de l'arbre de liaisons a permis de trouver **38** solutions (le circuit contient 8 cellules, et le nombre possibles de combinaisons et de 8! = 40320).

Si le nombre de cellules dans les vecteurs candidats est inférieur au nombre total de cellules, chaque cellule manquant est placée à côté de celle à laquelle elle est le plus fortement connectée.

d/ Évaluation de vecteurs candidats

La dernière partie de l'algorithme de placement de cellules, est le choix d'une solution parmi celles déterminées lors de l'exécution de la procédure antérieure. Pour chaque vecteur candidat, trois fonctions coût sont évaluées :

- longueur de la solution (en considérant les aboutements entre les cellules et les éventuelles cassures de diffusion),
- nombre de pistes nécessaires pour le routage,
- longueur du routage.

Le fait d'exécuter le routage pour chaque vecteur candidat peut paraître une tâche lourde. Toutefois, ceci est une condition nécessaire pour minimiser la hauteur de la bande (nombre minimum des lignes de routage) et la longueur des liaisons intra et inter-bandes.

De cette façon, la solution trouvée minimisera en même temps la longueur et la hauteur de la bande, en tenant compte non seulement des liaisons entre les cellules de la bande, mais aussi des liaisons entre les autres cellules du circuit.

3.7 ROUTAGE

Le routage est l'étape de génération des interconnexions des signaux équipotentiels du circuit. La qualité du routage est caractérisée par une surface occupée réduite. Ceci n'est possible que si la longueur des interconnexions entre les cellules est minimale. Il est donc important de bien placer les cellules. Dans notre générateur, le routage est effectuée au moment du positionnement des cellules.

Dans un environnement de **macro-cellules** le routage est normalement hiérarchisé due au grand nombre de connexions entre les blocs. Les étapes successives de cette approche sont:

- Le partitionnement de la région de routage. Les régions entre les sous-modules du circuit sont divisées en régions de formes régulières, généralement rectangulaires, sans obstacle à l'intérieur. Ces régions sont appelées canaux de routage.

- Le routage global. Le routage global détermine le chemin que le signal doit parcourir pour relier les noeuds équipotentiels. Il ne réalise pas les liaisons entre équipotentielles, mais distribue seulement les différents signaux par canal de routage, en attribuant la position relative de chacun sur les bords de chaque canal.

- Le routage détaillé. Les chemins reliant les noeuds présents dans les régions de routage sont créés par un routeur détaillé. Parmi les algorithmes de routage détaillé, la technique la plus couramment utilisée est le routage de canal. L'avantage de cette approche est que l'exécution est indépendante du routage des autres canaux. Ceci réduit sensiblement la complexité du problème.

Pour notre travail, génération de macro-cellules, le problème se limite au **routage des canaux**. Nous reviendrons sur la génération de canaux de routage et au routage global dans le chapitre qui traite d'environnement de macro-cellules.

Un critère important dans les approches où il n'y a pas de canaux de routage explicites est la routabilité complète, sans saturer les régions réservées à la réalisation des connexions. Nous n'avons pas ce problème, car la hauteur de chaque bande est variable, en fonction du nombre de pistes de routage nécessaires. Ainsi, le routage est toujours complet, sans intervention manuelle après la synthèse.

Un canal de routage peut avoir quatre listes de signaux : liste supérieure, liste inférieure, liste gauche et liste droite. Chaque signal doit être présent au moins deux fois dans l'ensemble des listes. La relation entre les signaux est représentée par deux graphes: graphe de contraintes horizontales et graphe de contraintes verticales.

Le graphe des contraintes horizontales est un graphe d'intervalles, non orienté. La construction du graphe est faite de la façon suivante: pour chaque signal '*s* ', nous déterminons un intervalle ***I(s)***, tel que l'extrémité gauche de cet intervalle commence dans la première colonne contenant s et son extrémité droite finit dans la dernière colonne contenant '*s* '. Tous les signaux contenus dans l'intervalle ***I(s)*** ajoutent une arête dans le graphe (les sommets étant les signaux). Dans la figure 3.30, par exemple, le signal 4 intercepte les signaux 2,3,1,5.

Le graphe de contraintes verticales est un graphe orienté contenant les signaux de la liste supérieure et inférieure se trouvant en face l'un de l'autre. Dans l'exemple, le signal 1 fait face au signal 3, 4 au 2, 3 au 2, 2 au 3 et 5, 5 au 4.

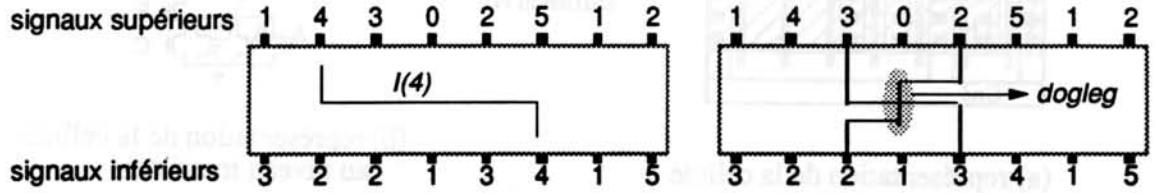


Figure 3.30- Exemple de canal de routage

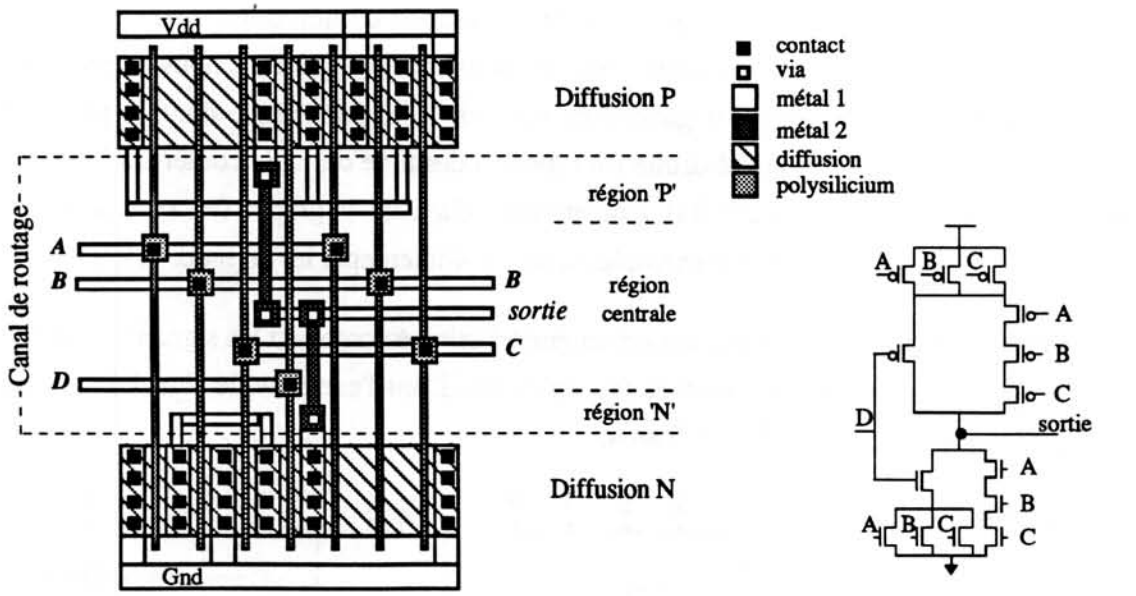
S'il y a des cycles dans le graphe de contraintes verticales, ils impliquent une augmentation du nombre de pistes nécessaires à la réalisation du routage. Dans l'exemple de la figure 3.30, il y a un cycle entre les signaux 2 et 3. Par conséquence, 3 pistes sont nécessaires pour connecter ces deux signaux, avec la réalisation des "*doglegs*".

3.7.1 Routage de canal pour les cellules "linear-matrix"

Chaque cellule "linear-matrix" peut avoir jusqu'à trois régions de routage: région 'N', région centrale et région 'P' (figure 3.31).

La région centrale permet le passage des connexions entre différentes cellules. Cette région a la caractéristique d'avoir toujours la liste des signaux supérieurs égale à la liste des signaux inférieurs, puisque les transistors ayant le même signal de grille sont placés dans la même colonne. Ceci implique l'absence de contraintes verticales, et par conséquence une simplification de l'algorithme à implanter.

Les régions 'N' et 'P' contiennent les connexions entre drains et sources équipotentielles d'une même cellule. Ces deux régions contiennent une seule liste de signaux, la liste supérieure pour la région 'N' et la liste inférieure pour la région 'P'.



(a) représentation de la cellule au niveau masques

(b) représentation de la cellule au niveau transistors

Figure 3.31 - Canal de routage d'une cellule "linear-matrix"

Dans la région centrale la connexion entre les signaux des transistors duaux est faite par une piste verticale de polysilicium. Des pistes verticales, en métal 2, relient les drains/sources de sortie vers la piste horizontale de routage contenant la sortie. Le nombre de pistes verticales en métal 2 est fonction du nombre de drains/sources connectées à la sortie de la cellule. Ces segments de métal 2 sont appelés "***stubs***" (ponts).

Dans les régions 'N' et 'P' nous utilisons des "stubs" entre le drain (ou la source) et la piste horizontale de routage (en métal 1). Dans notre exemple, les "stubs" sont absents, car ils ont déjà été filtrés. Le "filtrage" est une procédure exécutée après le routage, qui a pour objectif de réduire le nombre de trous de contacts, par le remplacement du métal 1 par le métal 2 (ou vice-versa).

Ainsi, le niveau métal 2 est utilisé verticalement pour connecter les plans duals des cellules (sorties) et pour le routage entre différentes bandes. Pour éviter les conflits d'utilisation intra et inter-bande, nous avons adopté la démarche suivante : l'espacement de la grille symbolique est multiplié par 2. Pour les positions paires on connecte les sorties de cellules et pour les positions impaires les liaisons entre les bandes.

Le niveau de diffusion n'est utilisé que pour réaliser les transistors, toute connexion dans ce niveau est interdite. La liaison avec l'alimentation du circuit est faite en métal 1. De multiples contacts sont placés dans les drains (ou sources) des transistors. Ce style d'implantation permet la réduction des parasites et de meilleures performances électriques du circuit.

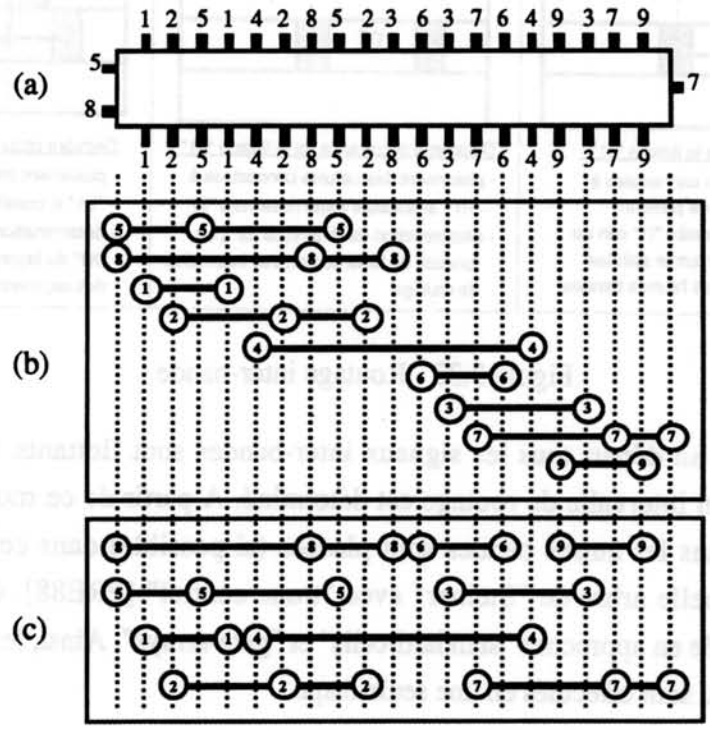
Comme nous avons dit précédemment, l'absence de contraintes verticales permet l'utilisation d'un algorithme plus simple. L'algorithme utilisé est appelé "left-edge". Il génère une solution avec un nombre minimal de pistes [HAS71]. Le nombre de pistes nécessaires à réalisation du routage est appelé *densité de connexion.*

Une remarque importante : au moment de la construction du graphe d'intervalles sont considérées non seulement les connexions internes, *mais aussi les connexions externes à la bande*. L'algorithme est le suivant:

procédure routage_canal()

{ 1 - Mise en ordre des signaux du canal à partir de l'extrémité gauche.
2 - Insérer dans le canal une nouvelle piste et placer sur cette piste le signal le plus à gauche.
3 - Insérer ensuite un nouveau signal dont l'extrémité gauche est la plus proche de l'extrémité droite du dernier signal inséré.
4 - Répéter 3 jusqu'à l'insertion de tous les signaux possibles dans la piste actuelle.
5 - S'il y a encore des signaux non placés, répéter à partir de l'étape 2.
}

La figure 3.32 montre l'exécution de cet algorithme sur un exemple. Cette procédure peut être améliorée par l'insertion de "doglegs". La méthode proposée par [SZY85] consiste à casser certaines pistes avant d'exécuter le routage, et ainsi réduire le nombre de pistes nécessaires. Toutefois, l'insertion de "doglegs" induit des parasites importantes, car il y a l'ajout de deux contacts et une piste verticale.



(a) canal de routage, (b) mise en ordre de signaux, à partir de l'extrémité gauche, (c) routage final

Figure 3.32 - Exécution du routage horizontal

3.7.2 Routage entre les lignes de cellules

Les lignes horizontales de cellules, appelées bandes, sont générées séparément. Lors du routage d'une bande '*b*', les signaux communs à '*b* ' et à d'autres bandes sont appelées signaux inter-bandes. Ces signaux ont deux classifications: *flottants* ou *fixes*

L'état initial de tous les signaux inter-bandes est flottant, sans contraintes de position. Quand un signal '*s*' est flottant, la génération de la première bande contenant '*s*' ne considère pas ce signal comme externe à la bande. Après le placement et le routage de la bande contenant '*s*', l'intervalle comprenant les extrémités gauche et droite de la connexion '*s*' est déterminée, afin de guider le placement d'autres cellules connectées à '*s*' dans les autres bandes (figure 3.33). À partir de ce moment le signal '*s*' est dit fixe.

Le placement des bandes contenant des signaux *fixes* prend en compte les intervalles de ces signaux. L'action, "considérer les signaux présents dans les bandes déjà générées", minimise la longueur des connexions.

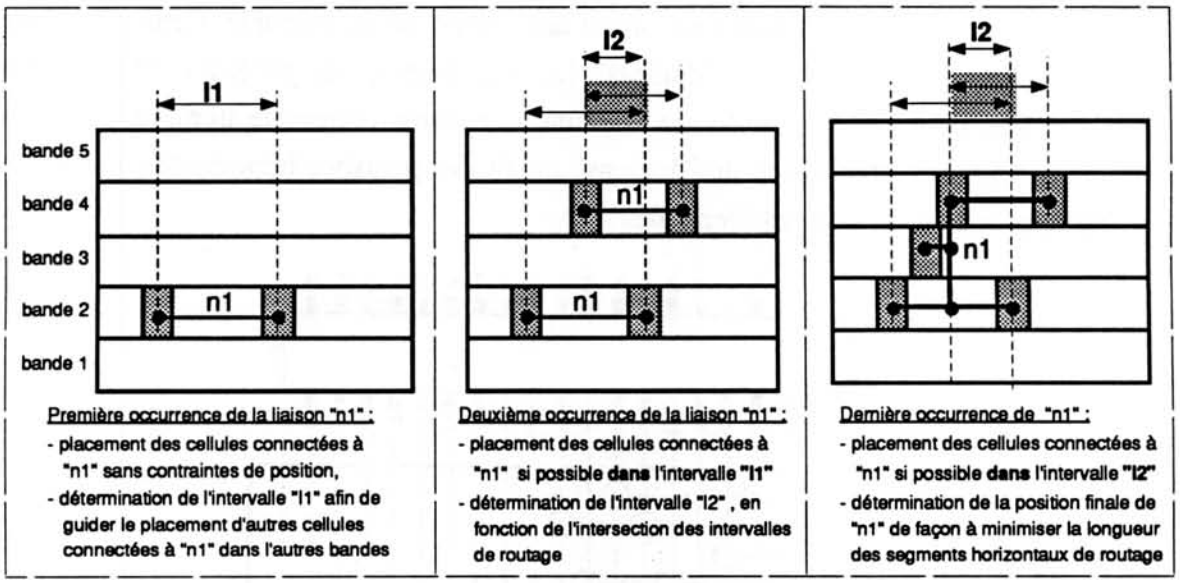


Figure 3.33 - Routage inter-bande

En résumé, au départ tous les signaux inter-bandes sont flottants. Dans la première occurrence de '*s*' un intervalle de routage est déterminé. A partir de ce moment, les cellules connectées à '*s*' dans les autres bandes sont placées (si possible) dans cet intervalle. Cette configuration s'appelle arbre de "Steiner" avec "tronc central" [PRE88]. Cette méthode est fréquemment utilisée en approches "standard-cells" et "gate-arrays". Ainsi, le routage horizontal et le routage vertical sont effectués en une seule étape.

La procédure générale pour la génération de la macro-cellule, en utilisant les algorithmes déjà présentés de positionnement et routage, est la suivante :

procédure génération du circuit (liste de cellules)

{ • Classer tous les signaux inter-bandes comme flottants.

• Fixer l'ordre de génération à partir des bandes avec le plus petit nombre de signaux inter-bandes (ceci pour avoir un nombre de contraintes de position réduit).

• Pour chaque bande, selon l'ordre fixé:

{ • Faire le placement : graphe de connexions, choix de graines, construction d'arbre de liaisons et évaluation des solutions candidates (routage, en considérant les signaux inter-bande fixées).

• Faire le routage de canal (régions 'N', centrale et 'P'), en utilisant "l'algorithme "left-edge".

• Fixer la coordonnée des signaux inter-bandes flottants

(on garde l'intervalle - x_{min} et x_{max} - du signal, afin de le connecter dans les autres bandes)

• Générer la description symbolique de la bande.

}

• Mettre les liaisons verticales de métal 2

• Faire les optimisations de routage

• Générer la description symbolique de chaque bande.

}

La hauteur de chaque bande correspond au nombre de pistes nécessaires au routage. Cette caractéristique, hauteur de bande variable, induit un routage à 100%, c'est à dire qu'il ne reste aucun signal à router manuellement après la génération du circuit.

3.7.3 Optimisations après le routage

La réalisation du routage impose l'utilisation d'au moins 2 couches technologiques: une couche pour les liaisons horizontales et une pour les liaisons verticales. Pour les relier nous utilisons des trous de contacts, qui sont source d'éléments parasites.

Nous présentons les optimisations faites sur le routage, qui ont pour objectif d'améliorer les performances électriques, et de réduire la surface du circuit. Les deux techniques utilisées sont :

- la réduction du nombre de trous de contacts (ou vias),
- le déplacement des pistes de routage de la région centrale vers les régions 'N' et 'P', en cas d'espace disponible.

La première optimisation, réduction du nombre de trous de contacts, est réalisée sur les "stubs" (segments verticaux de métal 2) et dans le canal de routage.

Un "stub" peut être implanté en métal 1, si aucune liaison horizontale en métal 1 ne le croise. Dans l'exemple (figure 3.34), tous les "stubs" des régions N et P ont été supprimés, car il n'y a pas de liaison en métal 1 qui les croisent. Pour les "stubs" qui relient les sorties de la

cellule au canal de routage, le "stub" 'P' reste en métal 2 car il y a des liaisons en métal 1 qui le croisent.

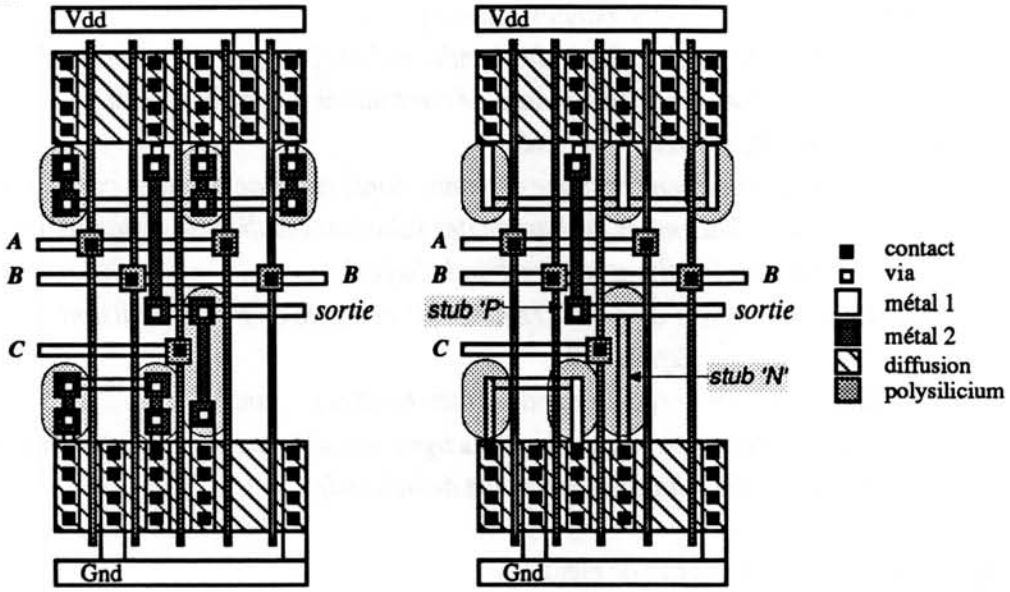


Figure 3.34 - Optimisation du Routage - suppression des "stubs"

La suppression de contacts dans le canal de routage est effectuée par le changement des segments de métal 1 qui ont le même type de contact dans ses extrémités, par des segments en métal 2 ou en polysilicium. Par exemple, si un segment de métal 1 a un "via" dans chaque extrémité, et qu'aucune liaison verticale en métal 2 ne le croise, ce segment peut être implanté en métal 2, et le contact de gauche éliminé (figure 3.35). Cette procédure est aussi appliquée sur le polysilicium. Dans le cas de circuits comportant de nombreuses portes de transmission, cette optimisation permet de supprimer un grand nombre de contacts.

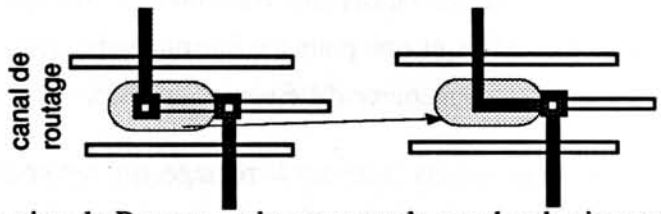


Figure 3.35 - Optimisation du Routage - changement de couches horizontales afin de supprimer contacts

La deuxième optimisation, le déplacement des pistes de routage, consiste à déplacer les liaisons de la région centrale de routage vers les régions 'N' et 'P'. Cette optimisation permet de diminuer le nombre de pistes nécessaires au routage de la bande, et ainsi de diminuer la surface finale du circuit. La mise en ordre de pistes de routage dans la bande doit prendre en compte la direction de signaux inter-bande. Par exemple, si un signal inter-bande connecte les bandes 1 et 2, ce signal dans la bande 1 doit être placé le plus haut possible. De même, dans la bande 2 il devra se trouver le plus bas possible, à fin de minimiser la hauteur de la ligne verticale de métal 2.

3.8 DESCRIPTION SYMBOLIQUE ET COMPACTAGE DE MASQUES

Le résultat de la génération topologique (génération des cellules, placement et routage) est un fichier symbolique, où les éléments constituant cette description sont :

- les transistors,
- les contacts,
- les liaisons,
- l'enveloppe,
- les connecteurs.

Les liaisons matérialisent les connexions et le choix des niveaux de routage (diffusion N, diffusion P, polysilicium, métal 1, métal 2). L'enveloppe définit la frontière de la macro-cellule. Les connecteurs caractérisent les communications entre l'intérieur et l'extérieur de la macro-cellule. Les connecteurs sont placées sur l'enveloppe.

La description symbolique est indépendante de la technologie. Les informations concernant la technologie contenues dans le fichier symbolique sont la largeur des liaisons et la dimension des transistors. La largeur des liaisons est fournie par le fichier de configuration du générateur, et les dimensions des transistors par le concepteur. Les règles technologiques (espacement, distance minimale, largeur minimale, par exemple) sont les paramètres d'outil du compacteur de masques.

La description symbolique [CAT90] contient 2 parties : la définition des éléments et la référence à ces éléments.

Partie définition

- Transistors: définition d'un symbole, *DS*, de numéro *100*, du type *transistor*, type N, avec une largeur de 7.5 µm et une longueur de 1.0 µm.

```
DS100        /Name=TRANSISTOR/
             /Type=MOSN/
             /W=7.50/
             /L=1.00/;
```

- Contacts: définition d'un symbole, *DS*, de numéro *101*, du type *contact*, qui connecte les niveaux diffusion N et métal 1, avec une largeur de 2.5 µm et une longueur de 7.5 µm.

```
DS101        /Name=CONTACT/
             /Type=DIFFNMET1/
             /WMet1=2.50/
             /LMet1=7.50/
             /W2=2.50/
             /L2=7.50/;
```

- Liaisons: définition d'une liaison, *DL*, de numéro *105*, du type *diffusion P*, avec une largeur de 7.5 µm et une priorité de 1250. La priorité précise l'ordre de traitement des liaisons par le compacteur PRINT. Les liaisons avec une valeur plus forte sont les premières à être traitées.

```
DL105        /Name=DIFFUSIONP/
             /Largeur=7.5/
             /Priorite=1250/;
```

- Connecteurs: définition d'un connecteur du type entrée/sortie.

```
DP0          /Name=PORT_E_S/ ;
```

- Enveloppe: définition de la frontière de la cellule. Dans l'exemple ci-dessous, entre les positions (0,0) et (78,28).

```
A 0 0 78 28 X 0 0 0 0;
```

Partie référence

- Transistors et contacts: dans le dessin symbolique ces éléments sont traités comme des points. La syntaxe est: *IS<numéro> <x> <y>*. Les deux exemples ci-dessous sont : la référence au transistor de type 100 dans la position (66,6) et la référence au contact de type 101 dans la position (42,6).

```
IS100 66 6;
IS101 42 6;
```

- Liaisons: dans le dessin symbolique les liaisons sont des fils, avec la largeur de la définition, ou avec une largeur fournie par le concepteur (par exemple, largeur de la piste d'alimentation). La syntaxe est : *IL<numéro> x1 y1 [largeur] x2 y2*. Les deux exemples suivants indiquent : la référence à la liaison de type 105, entre les coordonnés (64,22) et (68,22), et la référence à la liaison de type 8, avec une largeur de 5 µm.

```
IL105 64 22 68 22;
IL8 2 0 /5.00/ 2 28;
```

- Connecteurs: dans le dessin symbolique les connecteurs sont des éléments ponctuels, comme les transistors et les contacts. Les connecteurs peuvent avoir un nom, ce qui facilite le repérage des entrées et sorties dans le plan de masques. L'exemple ci-dessous montre un connecteur, dans la position (76,28), sur l'enveloppe, de nom 'vdd!'.

```
IP0 76 28 /vdd!/;
```

La figure 3.36 montre une partie de la description symbolique d'une cellule. La figure 3.37 montre le dessin symbolique correspondant à cette description.

```
DL0     /Name=METAL1/
        Largeur=1.5/
        /Priorite=50/;
DL1     /Name=METAL2/
        /Largeur=1.5/
        /Priorite=10/;
DL2     /Name=POLYSILICIUM/
        /Largeur=1.0/
        /Priorite=250/;
DL3     /Name=DIFFUSIONN
        /Largeur=1.0/
        /Priorite=1250/;
DL4     /Name=DIFFUSIONP/
        /Largeur=1.0/
        /Priorite=1250/;
DL8     /Name=METAL1/
        /Largeur=2.5/
        /Priorite=4000/;
DP0     /Name=PORT_E_S/ ;
DS10    /Name=CONTACT/
        /Type=POLYMET1/
        /WMet1=2.50/
        /LMet1=2.50/
        /W2=2.50/
        /L2=2.50/;
DS11    /Name=CONTACT/
        /Type=MET1MET2/
        /WMet1=2.50/
        /LMet1=2.50/
        /W2=2.50/
        /L2=2.50/;
DS12    /Name=CONTACT/
        /Type=BODY-N+/
        /WMet1=2.50/
        /LMet1=2.50/
        /W2=2.50/
        /L2=2.50/;
DS13    /Name=CONTACT/
        /Type=BODY-P+/
        /WMet1=2.50/
        /LMet1=2.50/
        /W2=2.50/
        /L2=2.50/;
DS14    /Name=CONTACT/
        /Type=MET1MET2/
        /WMet1=5.00/
        /LMet1=2.50/
        /W2=5.00/
        /L2=2.50/;
* Contact 1.00 2.50 ;
IL0 68 8 68 20;
IS11 69 8;
IL0 68 8 69 8;
IL1 30 8 30 20;
IL1 30 8 31 8;
IS11 30 20;
IS11 30 8;
IL1 20 8 20 18;
IL0 20 18 20 20;
IS11 20 18 /_16-X1/ ;
IS11 20 8;
* BOX 8 6 70 6 ;
* PWELL 8 2 70 6 ;
DS100   /Name=TRANSISTOR/
        /Type=MOSN/
        /W=7.50/
        /L=1.00/;
DS101   /Name=CONTACT/
        /Type=DIFFNMET1/
        /WMet1=2.50/
        /LMet1=7.50/
        /W2=2.50/
        /L2=7.50/;
DL102   /Name=DIFFUSIONN/
        /Largeur=7.50/
        /Priorite=1250/;
IS100 66 6;
IL102 64 6 68 6;
IS100 60 6;
IL102 58 6 62 6;
IS100 56 6;
IL102 54 6 58 6;
IS100 52 6;
IL102 50 6 54 6;
IS100 48 6;
...
IL102 62 6 64 6;
IS101 46 6 /_2-X1-X2/ ;
IS101 38 6;
IS101 16 6 /_10-X1-X3/ ;
...
IS101 68 6;
IS101 50 6 /_8-X1/ ;
IS101 30 6;
IS101 20 6;
* BOX 8 22 70 22 ;
* NWELL 8 22 70 26 ;
DS103   /Name=TRANSISTOR/
        /Type=MOSP/
        /W=7.50/
        /L=1.00/;
DS104   /Name=CONTACT/
        /Type=DIFFPMET1/
        /WMet1=2.50/
        /LMet1=7.50/
        /W2=2.50/
        /L2=7.50/;
DL105   /Name=DIFFUSIONP/
        /Largeur=7.50/
        /Priorite=1250/;
IS103 66 22;
IL105 64 22 68 22;
...
IL105 8 22 12 22;
IS104 62 22 /_7-X1-X2/ ;
IS104 28 22 /_15-X1-X3/ ;
IS104 20 22;
...
IS104 16 22;
IL0 50 20 66 20;
IL0 50 8 66 8;
IL0 62 22 62 24;
IL0 46 22 46 24;
IL0 38 22 38 24;
IL0 38 8 46 8;
IL0 46 6 46 8;
...
IL0 8 6 8 8;
IL8 64 22 64 26;
IL8 62 2 62 6;
...
IL8 24 22 24 26;
IL8 12 2 12 6;
IL0 68 20 68 22;
IL0 50 20 50 22;
IL0 30 20 30 22;
...
IL1 31 0 31 8;
IL1 69 0 69 8;
IL2 38 24 62 24;
IL2 8 24 28 24;
...
IL2 10 6 10 22;
IS11 7 12;
IS11 7 16;
IS11 7 14;
IS10 66 20;
IS10 66 8;
IS10 62 24;
...
IS10 20 24;
IS10 8 24;
IS12 64 26;
IS13 62 2;
IS12 42 26;
IS13 42 2;
IS12 34 26;
IS13 34 2;
IS13 28 2;
IS12 24 26;
IS13 12 2;
IL8 2 2 /2.50/ 62 2;
IL8 24 26 /2.50/ 76 26;
IP0 31 0 /carry/;
IP0 69 0 /sum/;
IL1 0 14 7 14;
IP0 0 14 /a/;
IL1 0 16 7 16;
IP0 0 16 /b/;
IL1 0 12 7 12;
IP0 0 12 /c/;
IP0 2 0 /gnd!/;
IP0 76 0 /vdd!/;
IP0 2 28 /gnd!/;
IP0 76 28 /vdd!/;
IS14 2 1;
IS14 76 1;
IS14 2 27;
IS14 76 27;
IL8 2 0 /5.00/ 2 28;
IL8 76 0 /5.00/ 76 28;
A 0 0 78 28 X 0 0 0 0;
```

Figure 3.36 - Description symbolique

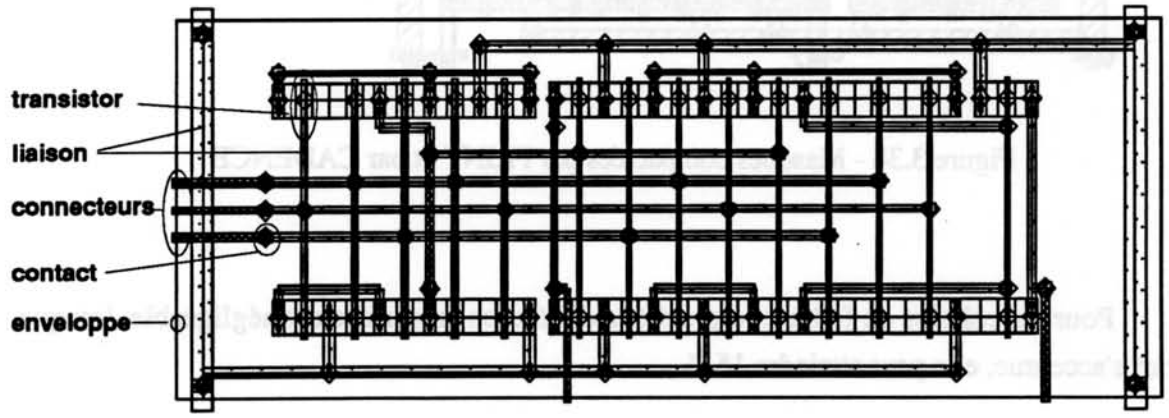


Figure 3.37 - Dessin symbolique

Cette description symbolique est convertie en une description au niveau masques en utilisant un compacteur. Nous utilisons, soit le compacteur PRINT [CAT90], développé au LIRMM, soit le compacteur de CADENCE. La conversion de la description symbolique vers la base de données CADENCE est faite par une procédure *skill* (annexe 3). La surface finale des macro-cellules traitées par le compacteur CADENCE est dans la plus parte des exemples plus petit. Ceci est dû à l'insertion de cassures ("jogs") dans les liaisons (figure 3.38).

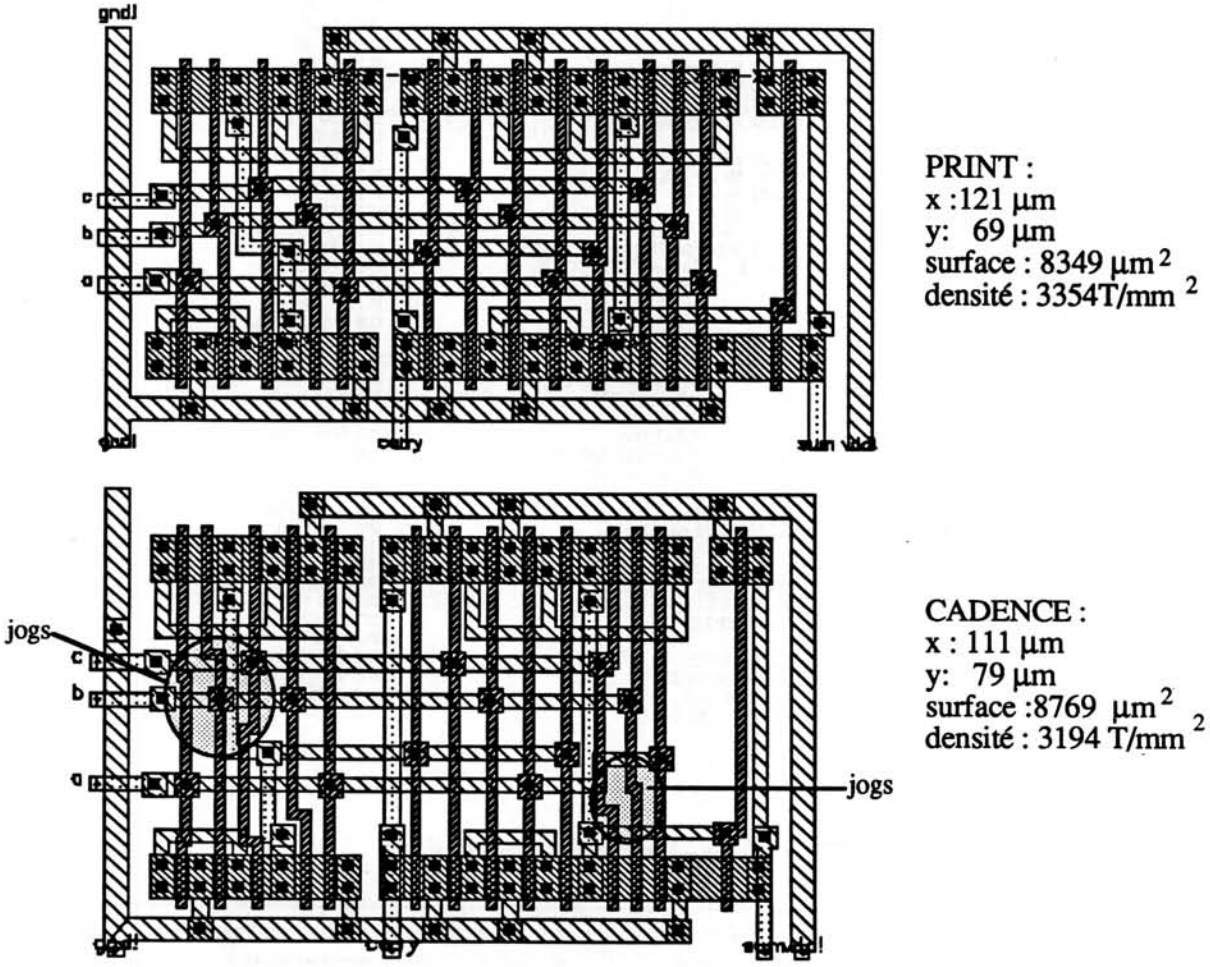


Figure 3.38 - Masques compactées par PRINT et par CADENCE

Pour les cellules de faible complexité la différence en surface est négligeable, lorsque l'écart s'accentue, elle peut atteindre 15 %.

3.8.1 Topologie des portes de transmission

Les portes de transmission sont générées selon une topologie pré-déterminée (figure 3.39). Cette topologie a l'avantage de permettre l'aboutement de portes de transmission avec d'autres portes de transmission ou cellules. S'il n'y a pas de cycles dans le routage (signal de grille P croisant le signal de grille N), il n'y a pas de perte de surface due à la non-dualité des signaux d'entrée de la porte de transmission. L'ordre des pistes de routage d'une bande est établi afin d'avoir le plus grand nombre possible de portes de transmission sans cycle. La figure 3.40 présente les masques d'une bascule D, avec 4 portes de transmission.

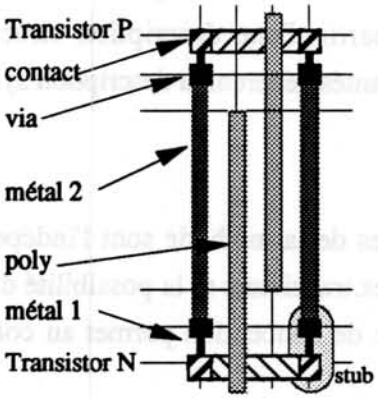


Figure 3.39 - Topologie d'une porte de transmission

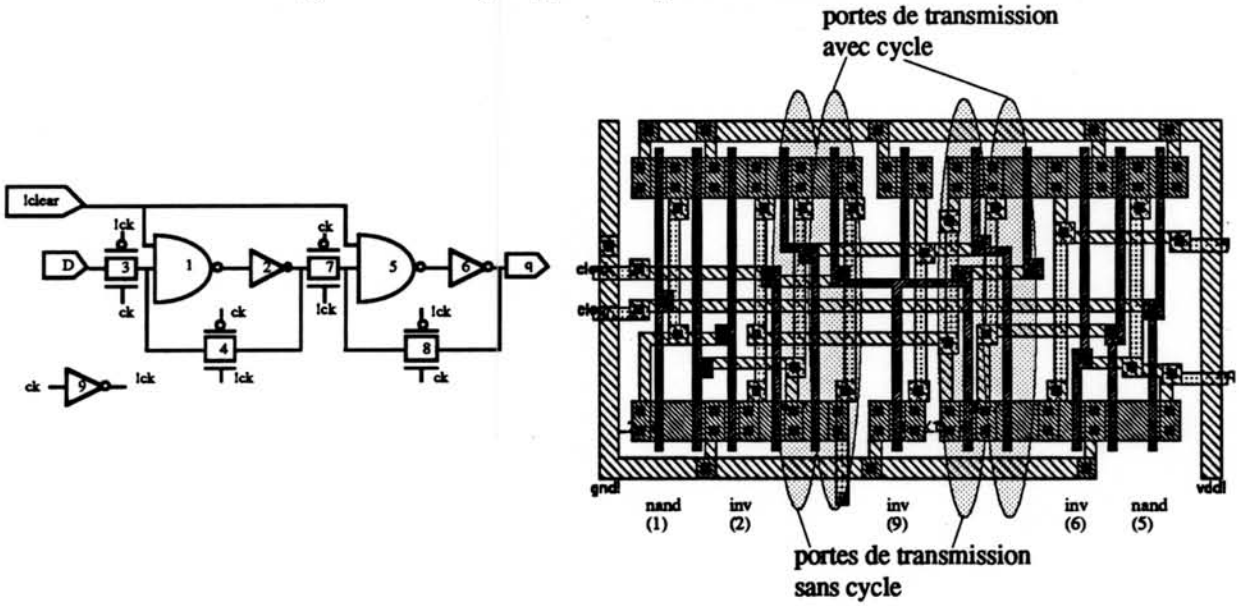


Figure 3.40 - Cellule avec portes de transmission (bascule D)

3.9 CONCLUSION

Dans ce chapitre nous avons détaillé la réalisation du générateur de macro-cellules. Pour cela nous avons utilisé des algorithmes de base, tels que la recherche du chemin d'Euler, la formation de groupes, les techniques de routage horizontal ("*left-edge*") et vertical (arbre de "*Steiner*"). Ils ont été adapté au style de dessin des masques choisie ("*linear-matrix*" multi-bandes) de façon à générer des macro-cellules optimisées en surface, avec un faible coût en éléments parasites. Ainsi, à partir d'une description structurelle (transistors ou portes) l'ensemble des algorithmes implantés génèrent la description symbolique du circuit.

Les principaux avantages de la méthode sont l'indépendance vis-à-vis des règles de dessin, le choix de la largeur des transistors et la possibilité d'utilisation de portes complexes dans le circuit. Cette souplesse de conception permet au concepteur d'avoir un circuit "sur mesure".

CHAPITRE 4

Environnement de macro-cellules

Etant donné les restrictions de temps CPU et de mémoire vive nécessaire au compactage de masques, les macro-cellules générées à l'aide de notre outil sont actuellement limitées à 5000 transistors. Afin de réaliser un circuit complet, contenant entre 30000 et 50000 transistors, les macro-cellules générées doivent être assemblées.

Nous présentons dans ce chapitre la méthode d'assemblage de plusieurs macro-cellules, générées à l'aide de notre outil de synthèse de masques, en utilisant un outil industriel de placement et de routage (CADENCE). Les étapes nécessaires à l'assemblage des macro-cellules sont les suivantes :

- *définition de la description d'entrée ;*
- *partition en macro-cellules ;*
- *création de la description interne à l'outil d'élaboration du plan directeur et de routage ;*
- *exécution du plan directeur ;*
- *génération des macro-cellules ;*
- *exécution du routage global.*

Ces étapes sont illustrées par la présentation d'un exemple d'un circuit réel, un filtre numérique.

4.1 DESCRIPTION D'ENTRÉE

L'utilisateur peut concevoir son circuit de quatre manières différentes :

- description haut-niveau (comportementale) ;
- schéma électrique, description graphique au niveau porte ;
- description textuelle au niveau porte ;
- description textuelle au niveau transistor.

La définition du circuit par une description de haut-niveau implique l'utilisation d'un outil de synthèse logique. La description comportementale a l'avantage de décrire l'algorithme que le concepteur veut implanter en silicium, sans restrictions structurelles. Le résultat de la synthèse logique est un schéma électrique, en utilisant les portes d'une bibliothèque spécifiée par le concepteur. Comme nous avons déjà montré (item 2.3), nous pouvons utiliser le langage *VERILOG* et l'outil *CADENCE HDL Synthetizer*.

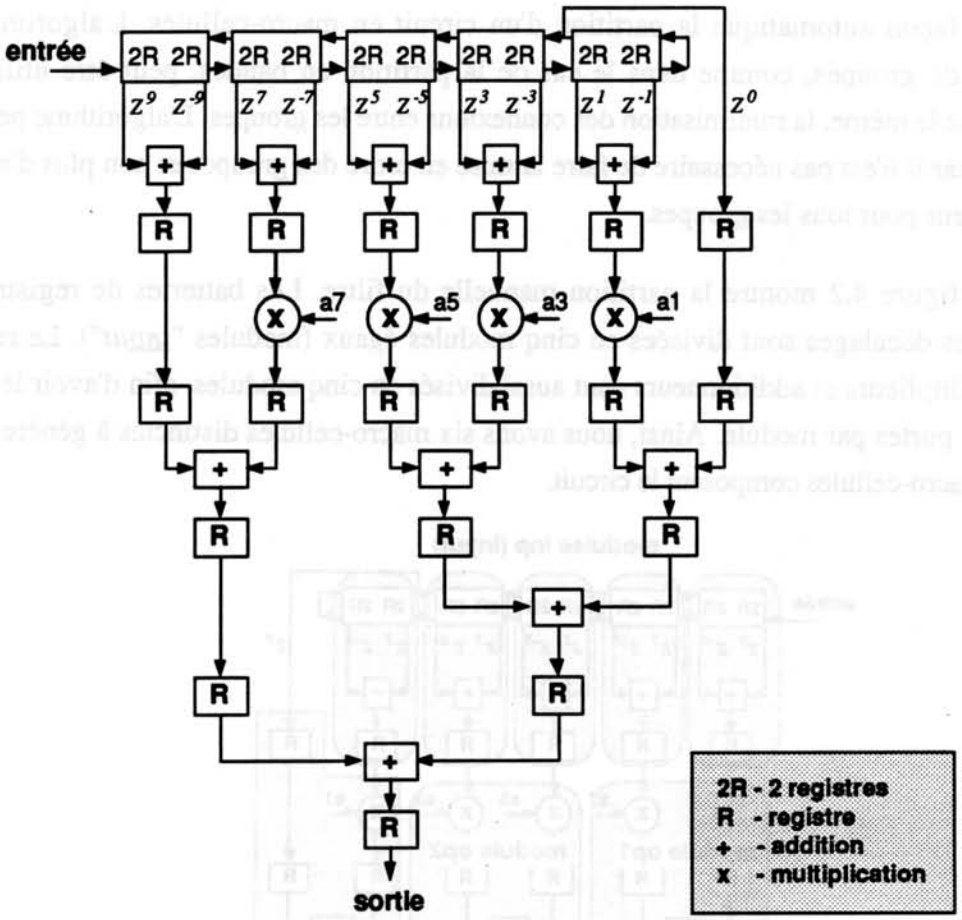
L'utilisation d'un schéma électrique, obtenu à partir de la synthèse logique ou défini graphiquement par le concepteur, implique l'existence d'un convertisseur qui génère une description textuelle au niveau portes, par exemple *HILO*, ou transistors, par exemple *SPICE*.

Le langage de description des circuits utilisé par notre système est la description *SPICE* **hiérarchisée**. Les opérations au niveau circuit, tels que l'assignation technologique et la partition en sous-modules, sont réalisées au niveau portes (portes élémentaires, bascules, additionneurs, etc...). Le niveau transistors est utilisé uniquement lors de la génération des macro-cellules.

Nous présentons dans la figure 4.1 l'architecture de l'exemple choisi pour illustrer l'assemblage des macro-cellules [COU92]. Le circuit est un filtre à réponse impulsionnelle finie, passe-bas, du dix-neuvième ordre à phase linéaire, destiné au traitement numérique des signaux vidéo. Le décalage temporel est effectué par une batterie de registres. Chaque registre décale les échantillons de z^{-1}. Le nombre de bits est de huit en entrée et de seize en sortie .

L'utilisation de multiplieurs classiques pose un problème de vitesse et de surface, car le nombre de couches logiques à traverser est important. Chaque coefficient est donc choisi de manière à diminuer la complexité des multiplieurs en minimisant le nombre d'étages. Ainsi, les multiplications sont réalisées par décalages et additions. Les termes "a0" et "a9" sont des décalages simples, sans additions. La partie critique est la multiplication des échantillons par le coefficient "a1", car deux additionneurs sont nécessaires pour la réaliser.

Le filtre a été défini par un schéma électrique, en utilisant l'outil *SOLO 2030*. À partir de ce schéma, une description au niveau logique dans le format *HILO* a été créée, afin de valider l'architecture par simulation logique. Ensuite, nous avons converti cette description au format *SPICE*, afin de pouvoir utiliser notre générateur de logique aléatoire (*TROPIC*). L'outil *OPUS* permet d'obtenir la description *SPICE* à partir d'un schéma électrique, sans passer par la description logique, à condition que les portes de la bibliothèque aient une description au niveau transistor.



$$F(z) = a0 + a1(Z^{1} + Z^{-1}) + a3(Z^{3} + Z^{-3}) + a5(Z^{5} + Z^{-5}) + a7(Z^{7} + Z^{-7}) + a9(Z^{9} + Z^{-9})$$

avec :

a0 = 0,5 (en binaire: 0,1 - decalage de um bit à droite)
a1 = 0,328125 (en binaire: 0,010101)
a3 = - 0,1171875 (en binaire: -0,0001111 ou -0,001+0,0000001)
a5 = 0,054687 (en binaire: 0,0000111 ou 0,0001-0,0000001)
a7 = - 0,0234375 (en binaire: -0,0000011 ou -0,000001+0,0000001)
a9 = 0,0078125 (en binaire: 0,0000001 - decalage de 7 bits à droite)

Figure 4.1 - Architecture du filtre

4.2 PARTITION EN MACRO-CELLULES

Selon la complexité du circuit, il est nécessaire de le diviser en macro-cellules. Actuellement, la complexité des macro-cellules est limitée à 1250 portes (environ 5000 transistors). Cette limitation est due aux contraintes d'occupation de mémoire vive et de temps CPU nécessaire au compactage des masques, ainsi que par le temps nécessaire à la validation temporelle de la structure générée.

La partition peut être manuelle ou automatique. Nous n'avons pas encore d'outil qui réalise de façon automatique la partition d'un circuit en macro-cellules. L'algorithme de formation de groupes, comme dans le cas de la partition en bandes, peut être utilisé car l'objectif est le même, la minimisation des connexions entre les groupes. L'algorithme peut être simplifié, car il n'est pas nécessaire de faire la mise en ordre des groupes et non plus d'avoir la même largeur pour tous les groupes.

La figure 4.2 montre la partition manuelle du filtre. Les batteries de registres qui réalisent les décalages sont divisées en cinq modules égaux (modules "*input*"). Le reste du circuit, multiplieurs et additionneurs sont aussi divisés en cinq modules, afin d'avoir le même nombre de portes par module. Ainsi, nous avons six macro-cellules distinctes à générer, et au total dix macro-cellules composent le circuit.

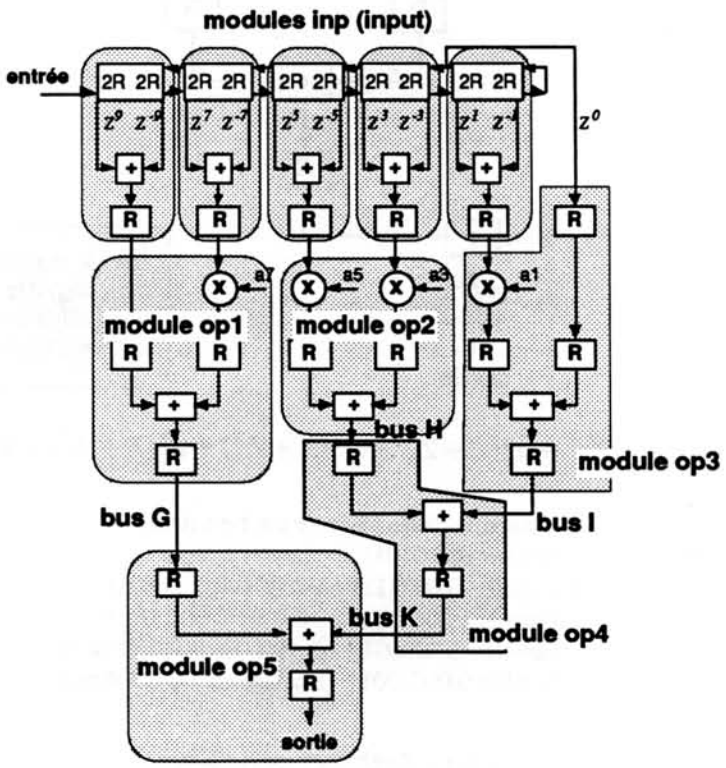


Figure 4.2 -Partition du filtre en macro-cellules

Le résultat de la partition est un fichier qui décrit l'assemblage de macro-cellules et les descriptions SPICE (niveau portes) de chaque macro-cellule. La figure 4.3 montre un fichier qui décrit l'assemblage de macro-cellules.

```
H exemple ES21.5 TROPIC /* header */

#   instances :  adders and flip-flops
I   dff      reg1   600    850    0
I   dff      reg2   800    850    0
I   dff      reg3   1000   850    0
I   adder    ad1    600    600    0
I   adder    ad2    800    600    0
I   adder    ad3    1000   600    0

#   signals of the circuit
N   input    a0     2    ad1.a     a0
N   input    b0     2    ad1.b     b0
N   input    a1     2    ad2.a     a1
N   input    b1     2    ad2.b     b1
N   input    a2     2    ad3.a     a2
N   input    b2     2    ad3.b     b2
N   input    cin    2    ad1.c     cin
N   logical  s3     2    ad2.c     ad1.cout
N   logical  s4     2    ad3.c     ad2.cout
N   output   cout   2    cout      ad3.cou
N   logical  s0     2    ad1.sum   reg1.D
N   logical  s1     2    ad2.sum   reg2.D
N   logical  s2     2    ad3.sum   reg3.D
N   input    clock  4    clock     reg1.clock  reg2.clock  reg3.clock
N   input    clear  4    clear     reg1.clear  reg2.clear  reg3.clear

N   output   q0   2   q0   reg1.q
N   output   q1   2   q1   reg2.q
N   output   q2   2   q2   reg3.q

# IOs of the circuit
C   a0     600    1100   N
C   a1     750    1100   N
C   a2     900    1100   N
C   b0     1050   1100   N
C   b1     1200   1100   N
C   b2     1350   1100   N
C   cin    0      600    N
C   clock  0      750    L
C   clear  0      900    L
C   q0     1500   600    R
C   q1     1500   750    R
C   q2     1500   900    R
C   cout   600    0      S
C   gnd!   750    0      S
C   vdd!   1050   0      S
# end of exemple
```

Figure 4.3 - Description décrivant l'assemblage de macro-cellules

Trois éléments composent cette description :

a/ Instances : *I <nom de la macro-cellule> <nom de l'instance> <x> <y> <orientation>*

La paire de coordonnées "*<x> <y>* " et l'orientation (0, 90, 180, 270) représentent un pré-placement de la macro-cellule. Par défaut, ces trois paramètres sont égaux à zéro. Le placement de la macro-cellule sera établi lors de la phase du plan directeur.

b/ Signaux : *N <type> <nom du signal> <n> <s1> <s1> ... <sn>*

avec *<type>* : input, output , logical (signal interne au circuit), clock ou *transparent*;
<nom du signal> ;
<n> : nombre de terminaux que le signal connecte;
<s1> <s1> ... <sn> : nom des terminaux.

Les noms des terminaux peuvent avoir une de ces deux syntaxes :

- *<nom de l'instance>.<nom du terminal de l'instance>*,
  exemple : "ad1.a", terminal "a" de l'instance "ad1";
- *<nom d'un plot d'entrée/sortie>*, exemple "a0".

Le signal du type "transparent" déclare les *jumpers* d'une macro-cellule. Un *jumper* est une connexion interne à une macro-cellule, entre au moins deux bords différents, sans être connectée aux transistors du module. Une connexion de ce type permet le routage sur les macro-cellules, et ainsi de réduire la longueur du routage et la surface occupée.

c/ Plots d'entrées/sorties: *C <nom> <x> <y> <orientation>*

Comme pour les instances, la paire de coordonnées "<*x*> <*y*>" et l'orientation (N, S, L, R) représentent un pré-placement du plot d'entrée/sortie. Par défaut, ces trois paramètres sont égaux à zéro. Le placement des plots d'entrées/sorties sera aussi établi lors de la phase du plan directeur.

La figure 4.4 illustre les modules qui composent la description de macro-cellules de la figure 4.3.

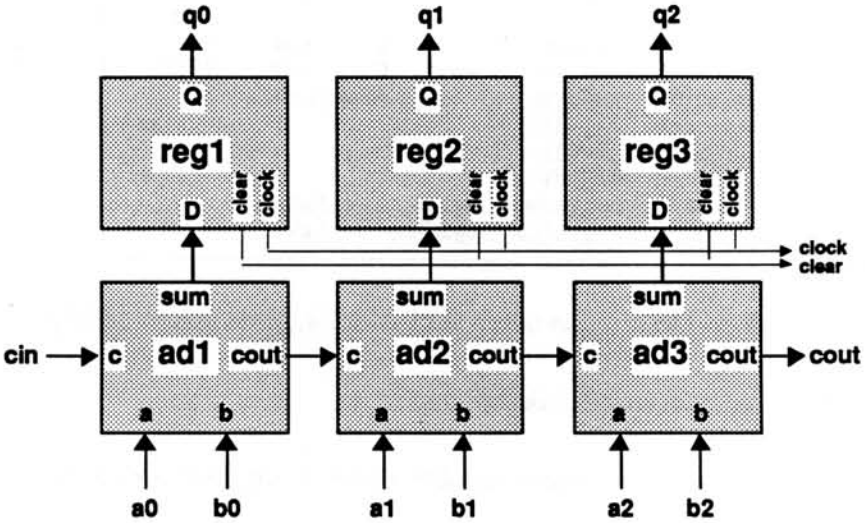


Figure 4.4 - Modules qui composent la description de la figure 4.3

4.3 DESCRIPTION DU CIRCUIT DANS L'OUTIL OPUS

Avant l'étape de routage global et du plan directeur, il faut créer la représentation du circuit dans l'outil de placement et de routage choisi. Nous décrivons les étapes nécessaires pour créer la représentation d'un circuit dans la base de données de l'environnement OPUS. Dans les autres environnements, comme OCTTOOLS, la démarche est analogue.

La représentation qui est utilisée pour le routage global et pour le plan directeur, dans l'environnement OPUS, est appelée "*autoLayout*". Les étapes nécessaires pour créer cette représentation sont décrites ci-dessous et illustrées dans la figure 4.5.

- Créer pour chaque macro-cellule la représentation "*abstract*". Cette représentation contient l'enveloppe du bloc, la position et la direction des plots d'entrées/sorties. Ces informations sont nécessaires au routage entre les différents blocs.

- Pour créer la description du circuit dans le format interne de l'outil, nous avons développé un convertisseur, écrit en langage "*skill*", qui génère toutes les représentations "*abstract*" et la représentation "*autoLayout*" du circuit, à partir la description d'assemblage de macro-cellules et des masques des modules.

- Exécuter l'étape du plan directeur (partie 4.4).

- Exécuter le routage global (partie 4.6).

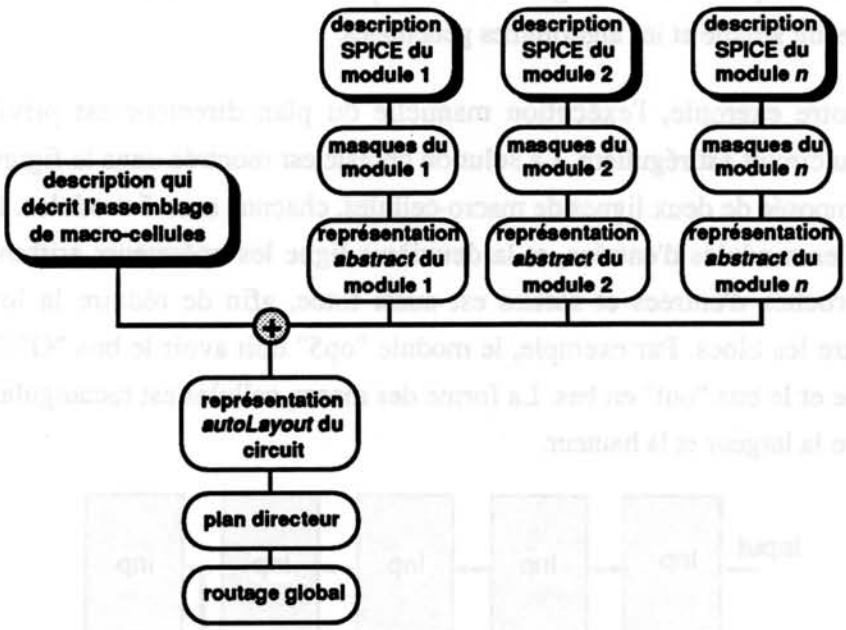


Figure 4.5 - Étapes de la création de la représentation "autoLayout"

4.4 EXÉCUTION DU PLAN DIRECTEUR

L'exécution du plan directeur est une opération très complexe. L'information fournie normalement à un outil du plan directeur est la suivante :

- la surface de chaque module (établie par des outils de prédiction de surface);

- la limite inférieure et supérieure du rapport entre la hauteur et la largeur du module (malléabilité);

- les connexions d'interfaces.

À partir de ces données, l'outil du plan directeur doit :

- établir la forme de chaque macro-cellule, en fonction de la malléabilité fournie;
- fixer dans quel bord les connexions d'interfaces sont placées;
- placer les macro-cellules, afin de minimiser la surface finale de la puce et/ou simplifier le routage global entre les modules.

Dans les algorithmes standard de placement de cellules, la forme des cellules et la position des plots d'entrées et de sorties sont déjà fixées. Ceci explique la complexité pour trouver une bonne solution pour le plan directeur d'un circuit quelconque, car le nombre de variables est très important. Les algorithmes les plus couramment utilisés sont donc les techniques de recuit simulé et les algorithmes génétiques.

Dans notre exemple, l'exécution manuelle du plan directeur est privilégiée, car l'architecture du circuit est régulière. La solution choisie est montrée dans la figure 4.6. Cette solution est composée de deux lignes de macro-cellules, chacune avec 5 modules. La première ligne contient les modules d'entrées, et la deuxième ligne les opérateurs arithmétiques. La position des broches d'entrées et sorties est aussi fixée, afin de réduire la longueur des connexions entre les blocs. Par exemple, le module "op5" doit avoir le bus "G" à gauche, le bus "K" à droite et le bus "out" en bas. La forme des macro-cellules est rectangulaire, avec un rapport 1:2 entre la largeur et la hauteur.

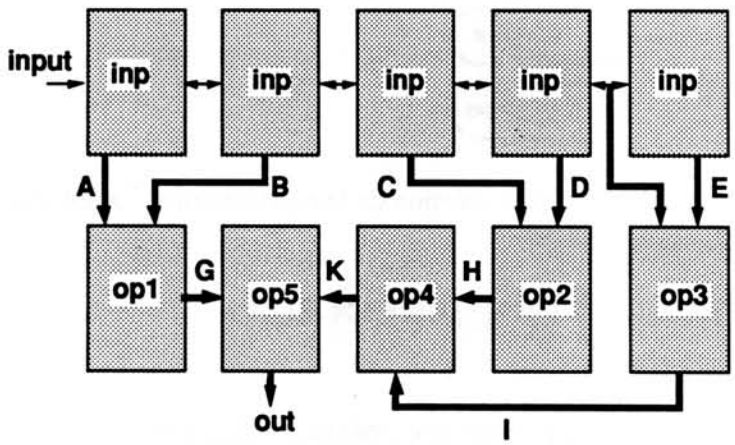


Figure 4.6 -Plan directeur du circuit filtre

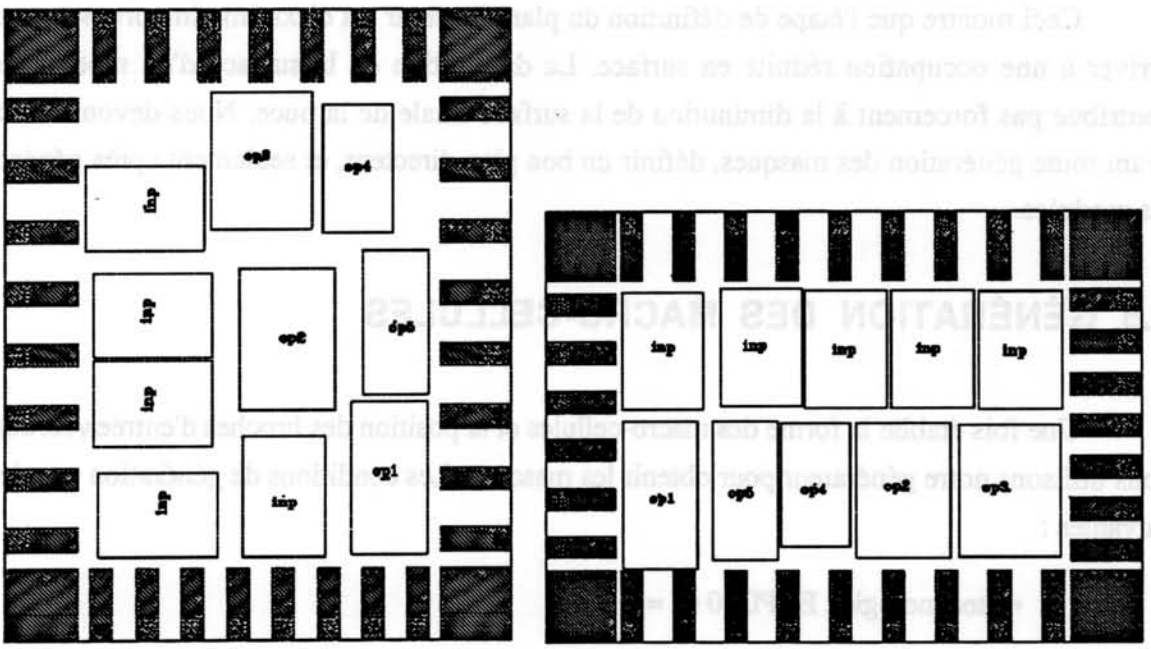
Deux outils de placement de macro-cellules ont été utilisés:

- "*puppy*", intégré dans le système OCTOOLS [OCT91];
- "*auto block place*", intégré dans l'environnement OPUS.

Les deux outils ont pour point commun l'utilisation d'un algorithme de recuit simulé. Ces deux algorithmes sont itératifs. Selon l'utilisation d'une fonction objectif ou d'une fonction de temps CPU, la solution de départ est améliorée.

La figure 4.7 montre les solutions manuelle et automatique (généré par OPUS) du plan directeur du circuit. Les principaux paramètres que le concepteur peut changer lors de l'exécution du plan directeur sont :

- temps d'exécution, plus ce temps d'exécution est grand, plus la probabilité d'obtenir de la part de l'algorithme une bonne solution est grande;
- solution à partir d'une solution initiale, si la solution de départ n'est pas fournie, l'algorithme la génère d'une manière aléatoire;
- la limite inférieure et supérieure du rapport entre la hauteur et la longueur du circuit, spécifie la forme que le circuit peut avoir;
- minimisation sous contraintes de la surface occupée et de la longueur du routage, un paramètre "α" et une fonction du type "$F=\alpha.S+(1-\alpha).R$" définissent le compromis entre la surface occupée (S) et le routage (R).



Solution automatique
(a) x=3003µm y=3680µm
Surface=11,05mm²

Solution manuelle
(b) x=3588µm y=2606µm
Surface=9,3503mm²

Figure 4.7 - Comparaison entre les solutions automatique et manuelle pour le plan directeur du circuit filtre

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

Le placement des plots peut être réalisé de deux façons distinctes :

- automatique, en utilisant une fonction de l'outil du plan directeur;
- manuel, à partir d'un fichier qui décrit le placement des plots (donne les meilleurs résultats) ou en déplaçant les plots.

L'outil du plan directeur utilisé, "*auto block place*" (OPUS), présente les problèmes suivants :

- la solution finale est fortement dépendante de la solution initiale et du temps CPU spécifié;
- plusieurs itérations sont nécessaires pour arriver à une "bonne" solution;
- le placement de modules peut générer des topologies qui empêchent la génération de canaux de routage, et ainsi des altérations manuelles sont souvent nécessaires.
- la surface finale, après le routage global, a été en moyenne 20% plus grande que la solution manuelle.

Ceci montre que l'étape de définition du plan directeur est d'extrême importance pour arriver à une occupation réduite en surface. La diminution de la surface d'un module ne contribue pas forcement à la diminution de la surface finale de la puce. Nous devons donc, avant toute génération des masques, définir un bon plan directeur, et seulement après générer les modules.

4.5 GÉNÉRATION DES MACRO-CELLULES

Une fois établie la forme des macro-cellules et la position des broches d'entrées/sorties, nous utilisons notre générateur pour obtenir les masques. Les conditions de génération sont les suivantes :

- technologie : ECPD10 (L = 1 µm)
- largeur des transistors : 10 µm (W_{MIN} = 1.25 µm)
- largeurs des rails d'alimentation : 5 µm
- compacteur des masques utilisé : "*VIRTUOSO COMPACTOR*" (OPUS)

- le nombre de cassures dans les connexions est limité à quatre, afin d'éviter l'effet "*échelle*" lors du compactage des masques;
- alignement des transistors.

Les résultats de la génération des macro-cellules sont illustrés dans le tableau 4.1.

module	nombre de transistors	nombre de cellules	nombre de TG	nombre de bandes	X (µm)	Y (µm)	Surface (mm^2)	Densité (Tr/mm^2)
inp	1038	280	164	8	495	701	0,3472	2990
op1	1226	307	140	11	456	892	0,4067	3014
op2	1336	311	104	10	569	824	0,4688	2850
op3	1410	333	148	9	601	805	0,4840	2913
op4	1094	266	136	10	420	750	0,3151	3472
op5	1074	254	124	11	397	845	0,3358	3198

Tableau 4.1 - Surface occupée et densité pour les macro-cellules du filtre (ECPD10)

Le nombre de bandes des modules est choisi de manière à :

- obtenir la même hauteur pour les modules "*op1*" à "*op5*";
- la somme de largeurs des modules "*op1*" à "*op5*" doit être à peu près cinq fois la largeur du module "*inp*".

Ces contraintes sont imposées par le plan directeur du circuit. Les différences en hauteur, 750 µm pour le module "*op4*" et 892 µm pour le module "*op1*", résultent d'espaces vides dans le circuit, la conséquence est une perte de surface (voir la figure 4.7.b). Nous avons essayé de générer les modules "*op1*" et "*op4*" avec différentes valeurs de nombre de bandes, cependant les valeurs retenus ont été les meilleures.

Les vérifications exécutées sur les circuits générés ont été :

- vérification des règles de dessin (DRC);
- extraction électrique;
- comparaison des netlists : comparaison entre la netlist SPICE d'entrée et le fichier généré par l'extraction électrique.

Le nombre total de transistors est de 11.330. La surface totale occupée par les macro-cellules est de 3,7464 mm^2. Ainsi, la densité moyenne de transistors par millimètre carré (W=10µm) est de 3024 T/mm^2. Nous avons aussi généré le filtre dans la technologie ECPD15 (L=1.6 µm, W=12µm). La densité moyenne obtenue pour cette technologie a été de 1300 T/mm^2. Le temps moyen pour la génération et le compactage de chaque module est d'environ 20 minutes (Sun Sparc 10).

4.6 ROUTAGE GLOBAL

Le routage global, comme expliqué dans le chapitre 3, est divisé en trois étapes:

- génération des canaux de routage;
- routage global;
- routage détaillé.

Parmi les paramètres que le concepteur peut choisir lors de l'exécution du routage, le plus important est le choix de la largeur et la priorité de routage de chaque signal. Par exemple, les signaux d'alimentation et d'horloge doivent être prioritaires, et routés si possible sans changement de niveau afin de réduire les éléments parasites.

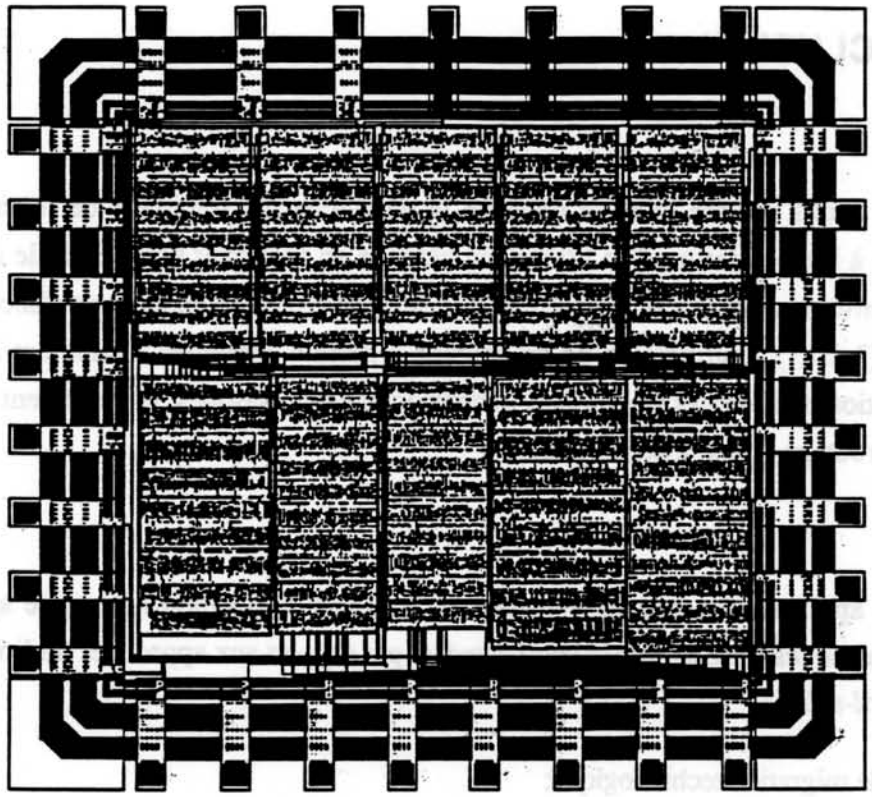
Dans notre exemple, après le routage global, la surface totale est de 9,3503 mm² (figure 2.9). La repartition de cette surface est la suivante :

• macro-cellules :	3,7464 mm²	(40 %)
• routage entre les macro-cellules:	1,0615 mm²	(11 %)
• plots d'entrées/sorties :	4,5424 mm²	(49 %)
TOTAL	9,3503 mm²	

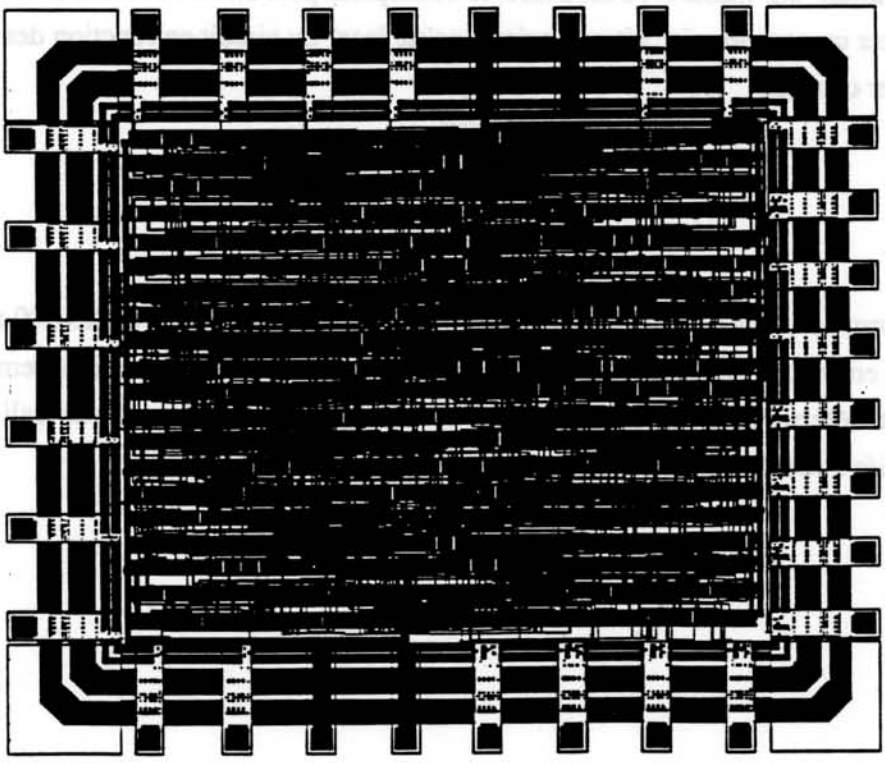
Avant d'exécuter l'extraction électrique, il est conseillé d'insérer les noms des signaux sur le routage, afin de les repérer plus facilement. Pour cela il existe aussi une procédure *skill* qui réalise l'insertion des noms automatiquement. Le résultat de l'extraction électrique sur ce circuit résulte en un fichier contenant 11330 transistors et 51578 capacités parasites.

La validation temporelle du circuit est effectuée par la simulation au niveau interrupteurs, en utilisant comme charge de sortie les capacités des plots d'entrée/sortie. Si l'architecture du circuit est régulière, comme dans le cas du filtre (architecture *pipeline*), le retard du circuit est le retard de la macro-cellule contenant le chemin critique. Dans le cas général, le concepteur doit trouver les vecteurs de test qui excitent le chemin critique du circuit.

Ce circuit a été implanté originalement en "standard-cell", dans une technologie 1,5 µm. La figure 4.8 montre les masques du filtre généré par notre approche et les masques de l'implantation standard-cell. Les surfaces (coeur du circuit, macro-cellules et canaux de routage) sont respectivement 9.80 mm² (3310 µm x 2962 µm) et 10,05 mm² (3486 µm x 2883 µm). La surface du circuit dans la technologie ECPD10 (1 µm) est de 4,8079 mm², soit 50% de gain par rapport à la technologie ECPD15 (1,5 µm) .



(a) Filtre généré par notre approche, surface 9,80 mm² (coeur)



(b) Filtre généré dans le style standard-cell, surface 10,05 mm² (coeur)

Figure 4.8 - Circuit filtre, généré par notre approche et dans le style standard-cell (ECPD15)

4.7 CONCLUSION

Nous avons illustré dans ce chapitre la démarche d'assemblage de macro-cellules. L'application à l'implantation d'un filtre a été obtenue en associant notre outil de synthèse de masques à l'environnement *CADENCE* (*EDGE* ou *OPUS*), grâce au langage d'accès à la base de données ("*skill*"). L'utilisation de ce langage a permis d'intégrer facilement la synthèse logique, l'édition de schémas électriques, le compactage des masques, le placement des macro-cellules et le routage global à notre système.

Cette application a mis en évidence les principaux avantages de notre approche de synthèse de masques (*linear-matrix multi-bandes*) par rapport aux approches traditionnelles du type "*standard-cells*" :

- facilité de migration technologique;
- la génération "*sur mesure*", c'est à dire, le concepteur peut choisir la taille des transistors et ainsi fixer un compromis "*vitesse/puissance/surface*" du circuit en fonction des contraintes du cahier charges.

Désormais, nous pouvons réaliser des circuits complexes (plus de 20.000 transistors), partitionnés en macro-cellules optimisées temporellement. Les performances temporelles du circuit étant déterminées par la simulation au niveau interrupteurs, après calibration des paramètres liés à la technologie.

CHAPITRE 5

Résultats de la génération de macro-cellules

L'objectif de ce chapitre est l'évaluation des paramètres qui déterminent la surface occupée et les performances électriques d'une macro-cellule générée selon le style d'implantation linear-matrix multi-bandes.

Le chapitre se décompose en 3 parties principales :

- *évaluation de la surface occupée,*
- *évaluation des performances électriques,*
- *comparaison entre l'approche utilisée dans notre générateur (TROPIC [MOR93]) et des approches industrielles.*

La première partie, évaluation de la surface occupée, comprend l'étude de six paramètres (sections 5.1 à 5.6):

- étude de la surface occupée, de la densité moyenne et du temps CPU, pour des exemples de complexité croissante (**nombre de transistors**);
- effet de l'évolution de la **technologie** sur la surface occupée;
- comparaison entre deux **compacteurs** de masques : PRINT [CAT90] et CADENCE™ [CAD89]. Le premier étant le compacteur développé au laboratoire, le deuxième un outil industriel;
- influence de la **largeur des transistors** sur la surface occupée;
- influence du **nombre de bandes** sur la surface occupée;
- comparaison des différents choix d'**architectures** pour implanter un circuit donné.

La deuxième partie, évaluation des performances électriques, comprend l'étude de quatre paramètres (sections 5.7 à 5.10):

- étude du rapport entre la **taille des transistors** et la **charge de sortie** sur les performances électriques du circuit, pour une technologie donnée;
- extension de l'étude antérieure à **différentes technologies**;
- effet du **dimensionnement** des transistors sur les performances électriques, utilisation d'une solution régulière et insertion d'étages tampons pour accélérer les performances d'un circuit;
- comparaison des performances des circuits implantés à l'aide de **portes élémentaires** (*nands*, *nors* et *inverseurs*) et de **portes complexes** (*AOIs*).

La dernière partie, comparaison entre l'approche utilisée dans notre générateur (TROPIC) et des approches industrielles, permettra de comparer notre méthode à l'approche standard-cells (section 5.11) et à un outil linear-matrix multi-bandes industriel (section 5.12).

Remarque : la technologie ciblée pour notre générateur est la technologie CMOS, utilisant deux niveaux de métal et un niveau de polysilicium. Dans ce chapitre nous utilisons cinq jeux de règles de dessin différentes : ECPD20, ECPD15, ECPD12, ECPD10 et ECPD07. Désormais, afin de faciliter la notation, nous appelons chaque ensemble de règles de dessin : ***"technologie"***.

5.1 ANALYSE DE LA SURFACE OCCUPEE

L'analyse de la surface occupée par les macro-cellules générées par notre outil a été effectuée sur un ensemble d'exemples de complexité croissante. L'exemple de plus faible complexité est l'inverseur (2 transistors). Celui avec le plus grand nombre de transistors est une partie d'un filtre numérique [COU92] (opérateur "*op1*", avec 1110 transistors). Le nombre de transistors dans une macro-cellule est limité par l'utilisation de mémoire vive, requise par le compacteur de masques utilisé.

Le tableau 5.1 présente les résultats de la surface occupée (en mm^2), la densité de transistors (en transistors/mm^2) et le temps CPU pour la génération et le compactage (en secondes - Sun Sparc 10). La technologie utilisée est ECPD15, tous les transistors étant dimensionnés au minimum (W=2µm et L=1.6µm). Le compacteur PRINT a été utilisé pour obtenir ce tableau. Ce compacteur n'insère pas de cassures dans les fils (polysilicium, métal 1 et métal 2).

CIRCUIT	bandes	nombre de trans.	nombre de portes	nombre de portes-transm	X µm	Y µm	Surface mm^2	Densité T/mm^2	Temps gén.(s)	Temps comp(s)	Temps total
inv	1	2	1	-	19	33	0,0006	3190	1,67	0,75	2,42
chaine	1	10	5	-	66	44	0,0029	3443	1,17	1,68	2,85
dff	1	22	5	4	119	62	0,0074	2982	3,1	3,7	6,8
mux2	1	24	4	8	179	58	0,0104	2312	2,0	5,2	7,2
adder-2 (AOI)	1	28	4	-	120	59	0,0071	3954	2,0	3,9	5,9
comp-1	1	36	6	4	206	55	0,0113	3177	4,5	6,8	11,3
adder-2 (gates)	1	40	10	-	194	65	0,0126	3172	6,1	6,8	12,9
shift4	2	88	20	16	271	147	0,0398	2209	17,8	24,3	42,1
compteur-4	3	118	24	22	252	223	0,0562	2100	67,6	38,5	106,1
shift8	3	176	40	32	383	248	0,0950	1853	86,7	75,8	162,5
ls161	3	206	37	16	401	289	0,1159	1777	41,3	84,6	125,9
compteur-8	4	238	48	46	379,5	347,5	0,1318	1805	42,2	128,1	170,3
alu4 (AOI)	3	262	39	-	458	290	0,1328	1972	58,3	109,1	167,4
comp-8	4	288	48	32	452	330	0,1492	1930	31,0	142,5	173,5
alu4 (gates)	5	432	101	-	471	420	0,1978	2184	100,2	239,0	339,2
ripple16	4	448	64	-	563	264	0,1486	3014	16,2	251	267,2
mult4	5	470	124	-	556	396	0,2202	2134	124,3	303,2	427,5
cla16	4	528	84	-	686	391	0,2682	1968	125,4	358,2	483,6
hdb3	6	570	118	-	540	575	0,3105	1836	761,4	433,0	1194,0
mult6	8	972	192	-	669	767	0,5131	1894	198,0	797,0	995,0
op1	8	1110	245	140	776	977	0,7582	1464	313,8	1276,0	1598,8
	Total :	6068				Total :	3.1896				

Tableau 5.1 - Surface occupée pour l'implantation de différentes macro-cellules (technologie ECPD15)

Ce tableau permet d'effectuer différentes remarques.

a/ L'augmentation de la surface en fonction du nombre de transistors n'est pas linéaire. Pour les macro-cellules de faible complexité la densité est plus élevée, car le nombre de connexions est assez faible. Au fur et à mesure que le nombre de connexions et de transistors augmente, le circuit généré devient moins dense.

b/ La densité moyenne des circuits sans porte de transmission est de 2078 T/mm^2 (3762 transistors pour une surface de 1,8018 mm^2). La densité moyenne des circuits avec portes de transmission est de 1662 T/mm^2 (2306 transistors pour une surface de 1,3875 mm^2). Cette différence est la conséquence de la dissymétrie introduite par les portes de transmission, (entrées non duales).

c/ La densité moyenne globale, pour les exemples présentés, est de 1902 T/mm^2 (6068 transistors pour une surface de 3,1896 mm^2). Cette densité permet d'avoir une estimation a priori de la surface que le générateur va utiliser lors de la génération de masques d'un circuit quelconque.

d/ Le temps CPU pour la génération de la description symbolique est fonction du nombre de transistors, du nombre de liaisons et du nombre de bandes. Par exemple, le circuit "*hdb3*" (570 transistors) a pris plus de temps CPU pour la génération de la description symbolique que le circuit "*op1*" (1110 transistors), pourtant plus complexe. Dans le premier circuit ("*hdb3*") le nombre de connexions entre les cellules est beaucoup plus important.

e/ Le temps CPU pour le compactage de masques est fonction du nombre d'éléments à traiter. Au-delà d'un certain nombre de transistors, c'est lui qui impose le temps total.

5.2 SURFACE OCCUPEE - DIFFERENTES TECHNOLOGIES

Dans les approches traditionnelles, du type "*standard-cells*", le changement de technologie (règles de dessin), implique d'attendre la mise à jour de la bibliothèque. Dans notre générateur cette opération est très simple : il suffit de changer le fichier qui décrit les règles de dessin (la taille des transistors étant définie dans ce fichier ou fournie par le concepteur).

La figure 5.1 montre l'évolution de la surface occupée, pour le circuit additionneur 2 bits, pour les différentes technologies utilisées. Les tableaux 5.2 et 5.3 présentent les résultats de la surface occupée et de la densité de transistors pour différentes macro-cellules et différentes technologies.

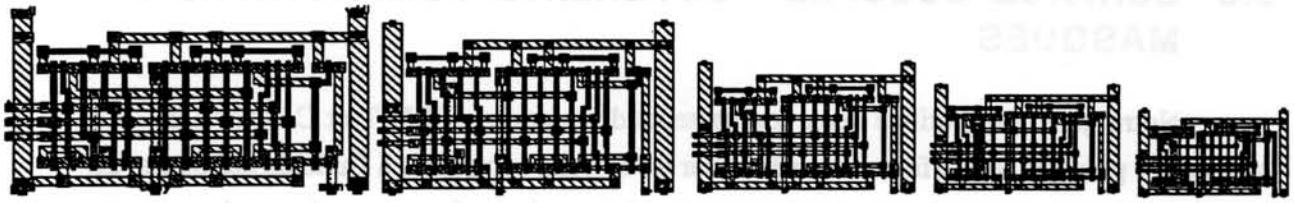


Figure 5.1 - Évolution de la surface occupée, pour les différentes technologies (cellule additionneur)

circuit	bandes	TR#	ECDM20 w = 3 µm	ECPD15 w = 2 µm	ECPD12 w = 1,5 µm	ECPD10 w = 1,25 µm	ECPD07 w = 1 µm
dff	1	22	0,0101	0,0077	0,00462	0,00336	0,00215
adder-2 (AOI)	1	28	0,0107	0,0083	0,00488	0,0040	0,00217
ls161	3	206	0,1528	0,1104	0,0653	0,04632	0,0309
alu4 (AOI)	3	262	0,1552	0,1097	0,0635	0,04669	0,0299
ripple16	4	448	0,2324	0,1805	0,1017	0,06702	0,0431
cla16	4	528	0,3781	0,2839	0,1615	0,1078	0,0787
	Total :	1494	0,9393	0,7005	0,40150	0,27519	0,18692

Tableau 5.2 - Surface occupée pour différentes technologies (en mm^2)

circuit	bandes	TR#	ECDM20 w = 3 µm	ECPD15 w = 2 µm	ECPD12 w = 1,5 µm	ECPD10 w = 1,25 µm	ECPD07 w = 1 µm
dff	1	22	2183	2836	4759	6542	10209
adder-2 (AOI)	1	28	2604	3380	5735	8244	12916
ls161	3	206	1348	1865	3152	4447	6659
alu4 (AOI)	3	262	1675	2370	4093	5509	8667
ripple16	4	448	1927	2482	4409	6684	10397
cla16	4	528	1396	1860	3269	4897	6708
	Total :	1494	$\bar{d}$ = 1588	$\bar{d}$ = 2130	$\bar{d}$ = 3716	$\bar{d}$ = 5421	$\bar{d}$ = 7982

densité moyennne = $\bar{d} = \Sigma$ surfaces + Σ transistors

Tableau 5.3 - Densité pour différentes technologies (en transistor/mm^2)

L'évolution des règles de dessin n'est qu'une réduction, linéaire ou non, dans les contraintes d'espacement et de largeurs minimales. Par exemple, entre la technologie ECPD15 et ECPD07 toutes les règles de dessin ont été divisées par deux. Par conséquence, la surface a été divisée par quatre (surface = (longueur + 2) * (hauteur + 2) = longueur * hauteur + 4).

Le générateur "*Lib*" [HSI91] permet d'obtenir une densité moyenne de 3736 T/mm^2, pour une technologie 1.2 µm, sur un ensemble d'exemples de faible complexité (jusqu'à 48 transistors). Le générateur "*iCoach*" [CH88] permet d'obtenir, pour une technologie 3 µm, une densité moyenne de 605 T/mm^2 (circuit ALU) et 515 T/mm^2 (circuit ayant 978 transistors).

5.3 SURFACE OCCUPEE - DIFFERENTS COMPACTEURS DE MASQUES

Notre générateur utilise deux compacteurs de masques : PRINT et CADENCE™. Le tableau 5.4 présente les surfaces occupées pour les circuits compactés à l'aide de ces outils. La dernière colonne montre le rapport (pourcentage) entre la surface occupée par les circuits compactés par CADENCE™ et PRINT.

CIRCUIT	TR#	Surface mm^2 PRINT	Densité T/mm^2	Surface mm^2 CADENCE™	Densité T/mm^2	surface CADENCE / surface PRINT
inv	2	0,0006	3190	0,00073	2742	1,22
chaine	10	0,0029	3443	0,0029	3487	1,00
dff	22	0,0074	2982	0,0078	2837	1,05
mux2	24	0,0104	2312	0,0118	2033	1,13
adder-2 (AOI)	28	0,0071	3954	0,0085	3298	1,20
comp-1	36	0,0113	3177	0,0134	2680	1,19
adder-2 (gates)	40	0,0126	3172	0,0130	3076	1,03
shift4	88	0,0398	2209	0,0360	2442	0,90
compteur-4	118	0,0562	2100	0,0506	2333	0,90
shift8	176	0,0950	1853	0,0907	1941	0,95
ls161	206	0,1159	1777	0,1105	1865	0,95
compteur-8	238	0,1318	1805	0,1239	1921	0,94
alu4 (AOI)	262	0,1328	1972	0,1199	2185	0,90
comp-8	288	0,1492	1930	0,1398	2059	0,94
alu4 (gates)	432	0,1978	2184	0,1785	2420	0,90
ripple16	448	0,1486	3014	0,1805	2482	1,21
mult4	470	0,2202	2134	0,2023	2323	0,92
cla16	528	0,2682	1968	0,2838	1861	1,06
hdb3	570	0,3105	1836	0,2909	1959	0,94
mult6	972	0,5131	1894	0,4614	2106	0,90
op1	1110	0,7582	1464	0,6411	1731	0,85
op24 *	2220	-	-	1,4935	1486	-
	6068	3,1896	1902	2,96803	2044	0,93

**n'est pas considéré pour le calcul de la densité moyenne*

Tableau 5.4 - Surface occupée pour différentes macro-cellules, compactées par les outils PRINT et CADENCE™ (technologie ECPD15)

Les avantages du compacteur CADENCE™ sont :

- insertion de cassures ("*jogs*") dans les fils, ce qui a pour conséquence une réduction moyenne de 7% dans la surface finale,
- simplicité pour changer les règles de dessin,
- environnement graphique,
- temps CPU réduit,
- possibilité de compacter des macro-cellules plus complexes (plus grand nombre de transistors), par exemple, 2220 transistors.

L'avantage du compacteur PRINT réside dans la simplicité d'utilisation. Il peut être intégré directement dans le générateur, car nous possédons les sources du logiciel. Par contre l'utilisation de mémoire vive par PRINT est très importante, la complexité est limitée autour de 1200 transistors et le changement des règles de dessin est très difficile.

L'insertion de cassures sur des lignes parallèles proches, comme dans le cas des lignes de polysilicium, peut entraîner une augmentation importante de la hauteur de la macro-cellule (effet "échelle", voir la figure 3.38). Ceci explique pourquoi les circuits générés avec une seule bande, compactés par CADENCE™, ont une surface plus importante.

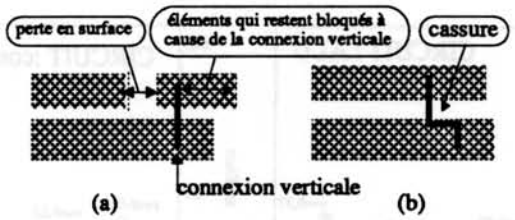


Figure 5.2 - Effet de "blocage", due à la non-insertion de cassures dans le compactage

Si le compacteur n'insère pas des cassures dans les fils, les connexions verticales peuvent "bloquer" le compactage. Par exemple (figure 5.2.a), si la position de la connexion verticale est fixée, les déplacements des éléments à sa droite sont limités par sa position. Par contre, si l'insertion de cassures est permise, ce blocage n'arrive pas (figure 5.2.b), et le compacteur peut mieux compacter les bandes dans la direction horizontale.

Les cassures sur les fils ("*jogs*") sont donc nécessaires pour les liaisons entre différentes bandes, en métal 2, car elles permettent d'éviter les blocages induits par le routage vertical.

5.4 SURFACE OCCUPEE - DIFFÉRENTES LARGEURS DE TRANSISTORS

Dans le style "*linear matrix*", pour une implantation régulière (même largeur des transistors pour tous les transistors du circuit), l'augmentation de surface est linéaire en fonction de la largeur des transistors (figure 5.3).

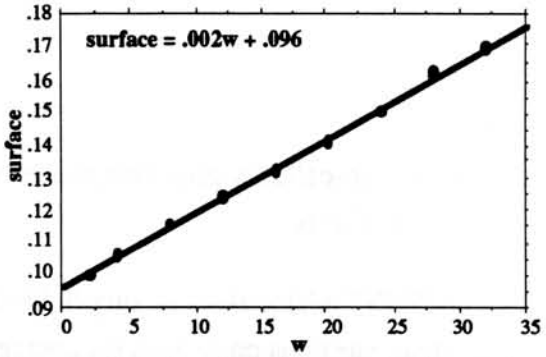


Figure 5.3 - Surface en fonction de la largeur des transistors (circuit ALU)

5.5 SURFACE OCCUPEE - DIFFÉRENTS NOMBRES DE BANDES

Les courbes ci-dessous (figure 5.4) présentent l'occupation de surface des circuits ALU et compteur 4 bits en fonction du nombre de bandes. La figure 5.5 présente les masques du circuit ALU généré avec différents nombres de bandes.

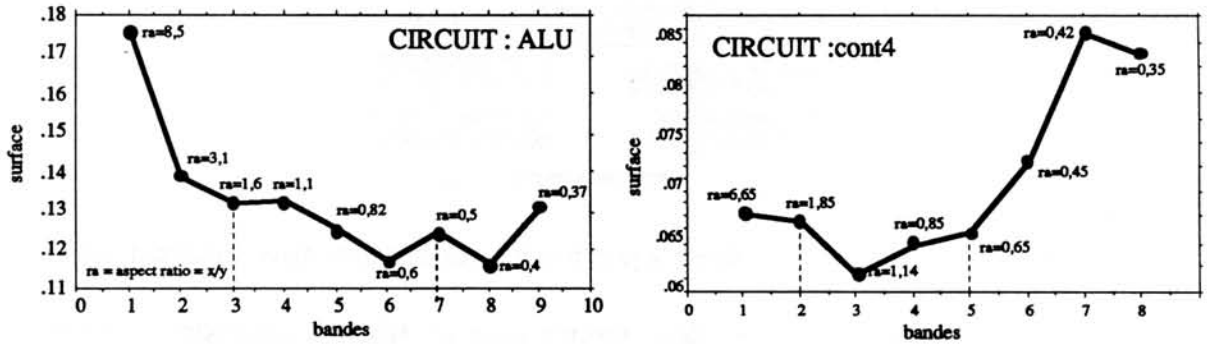


Figure 5.4 - Evolution de la surface occupée en fonction du nombre de bandes, pour le circuit ALU (260 transistors) et pour un compteur 4-bits (118 transistors)

A un nombre réduit de bandes correspond une augmentation de surface (par exemple, le circuit ALU avec 1 bande). Ceci est dû au fait que la plupart des connexions sont réalisées horizontalement. Au fur et à mesure que le nombre de bandes augmente, le routage est partagé entre les connexions horizontales et verticales : l'occupation de surface est donc optimisée. Au-delà d'un certain nombre de bandes, la surface augmente de nouveau, cette fois par le routage vertical (par exemple, le circuit compteur avec 7 bandes).

Le paramètre "rapport d'implantation" est défini comme le rapport entre la longueur et la hauteur de la macro-cellule. A partir des courbes, nous remarquons que si le "rapport d'implantation" est compris entre 0.5 et 2 (*hauteur = 2 * longueur* et *hauteur = 0.5 * longueur*), la surface occupée est optimisée.

Ainsi, en connaissant la densité moyenne pour une technologie donnée et le "rapport d'implantation" qui optimise la surface occupée, il est possible de prévoir la forme et la surface de la macro-cellule avant la génération. Ceci permet, lors de l'exécution du plan directeur d'un circuit, de faire un pré-placement des macro-cellules, et ainsi de définir la position des broches d'entrées/sorties et de prévoir l'espace nécessaire au routage entre les macro-cellules.

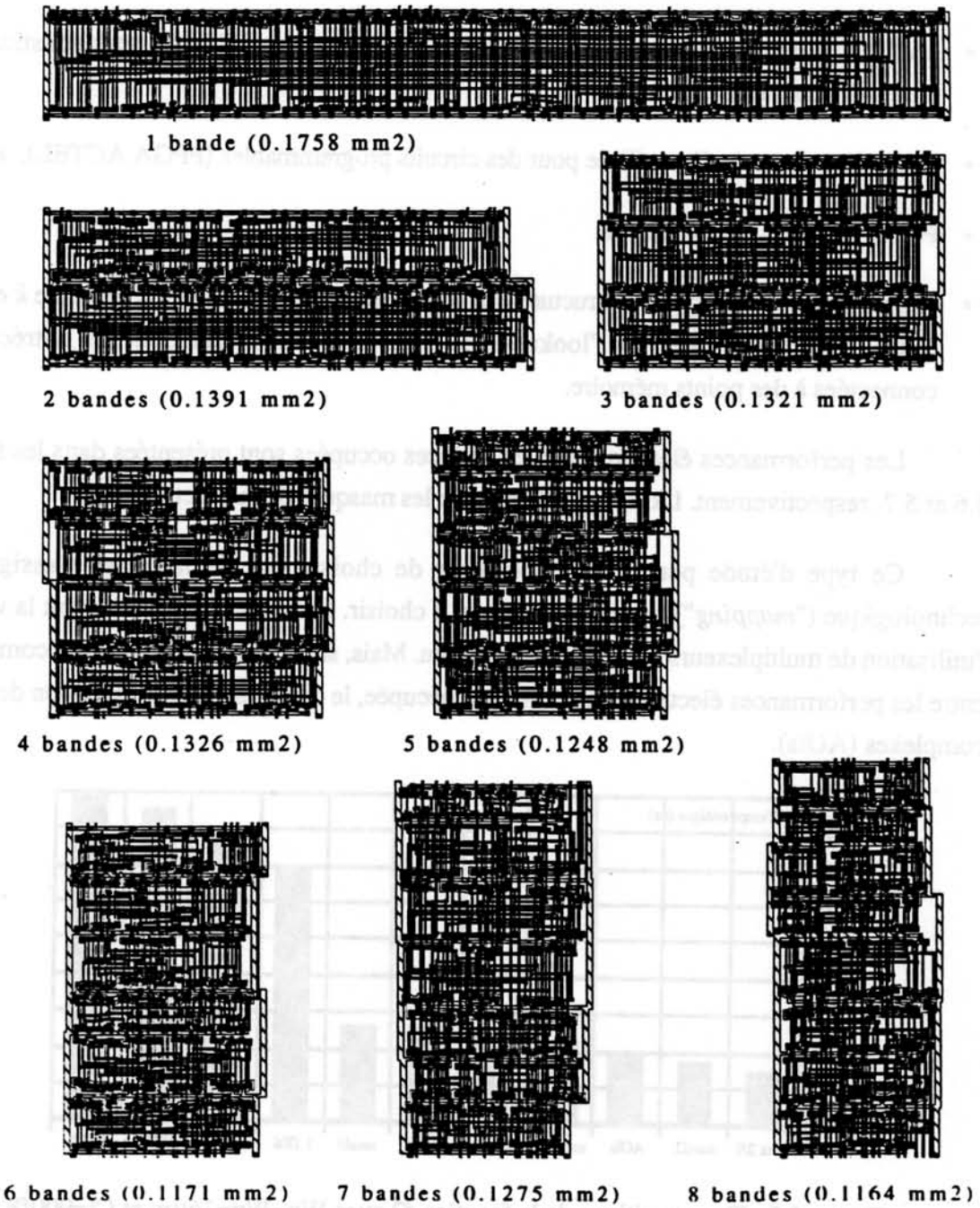


Figure 5.5 - Circuit ALU généré avec différents nombres de bandes

5.6 SURFACE OCCUPEE- DIFFÉRENTES ARCHITECTURES

[ROB94] compare les performances électriques et les surfaces nécessaires à l'implantation des fonctions logiques dans les circuits programmables. L'objectif est de comparer les performances des différentes architectures.

Nous présentons une de ces fonctions :

$$f2 = a.b + f.c.d + \overline{f}.b.e + \overline{b}.\overline{c}.d \qquad \text{(6 variables)}$$

Les architectures choisies pour implanter cette fonction logique sont :

- Multiplexeurs deux vers un (*mux2/1*) et quatre vers un (*mux4/1*). Ils sont constitués de portes de transmission.
- Multiplexeur particulier utilisée pour des circuits programmables (FPGA ACTEL), *muxA*.
- Portes nand à 2, 3 et 4 entrées.
- "Lookup tables". Ce type de structure permet de mémoriser une table de verité de *k* entrées (*ltk3*, *ltk4*, *ltk5* et *ltk6*). La "lookup table" est un multiplexeur dont les entrées sont connectées à des points mémoire.

Les performances électriques et les surfaces occupées sont présentées dans les figures 5.6 et 5.7, respectivement. La figure 5.8 présente les masques de la fonction f2.

Ce type d'étude permet au concepteur de choisir lors de la phase d'assignation technologique ("*mapping*") le type de structure à choisir. Si le critère est seulement la vitesse, l'utilisation de multiplexeurs est la meilleure option. Mais, si le critère est le meilleur compromis entre les performances électriques et la surface occupée, le choix est donc l'utilisation de portes complexes (AOIs).

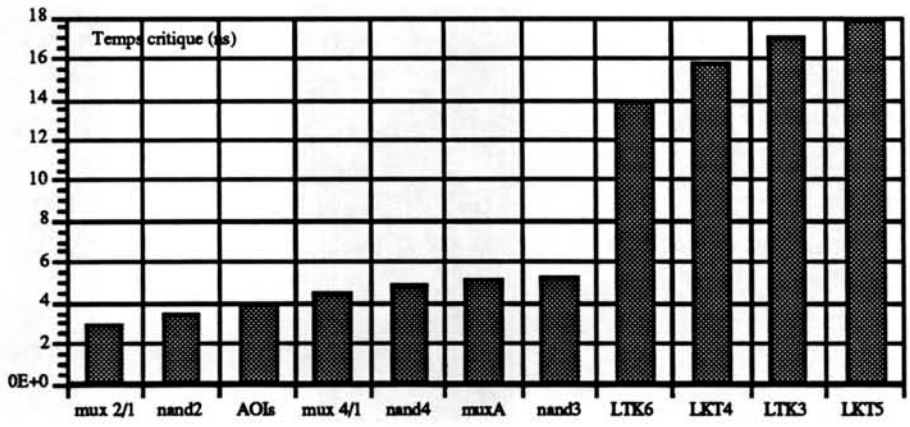


Figure 5.6 - Temps critique de la fonction f2 avec Wn=Wp=16µm et Cl=900fF

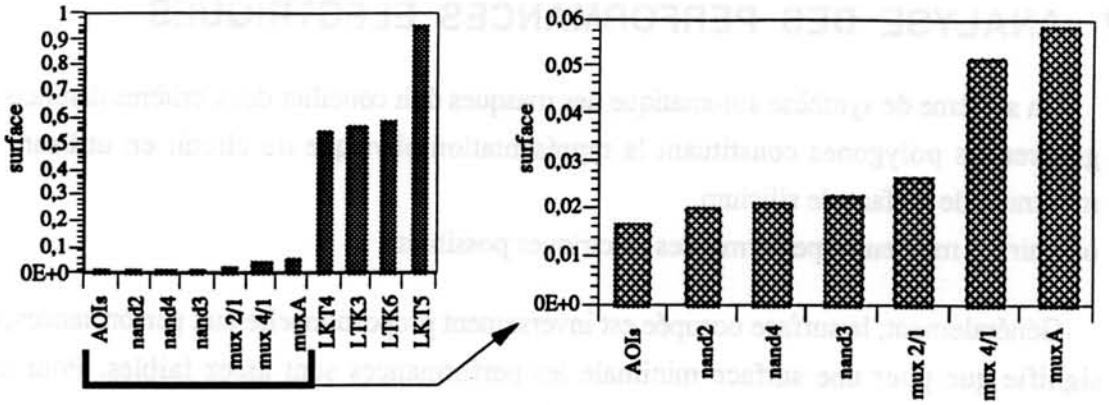


Figure 5.7 - Surface nécessaire pour l'implantation de la fonction f2, pour différentes architectures

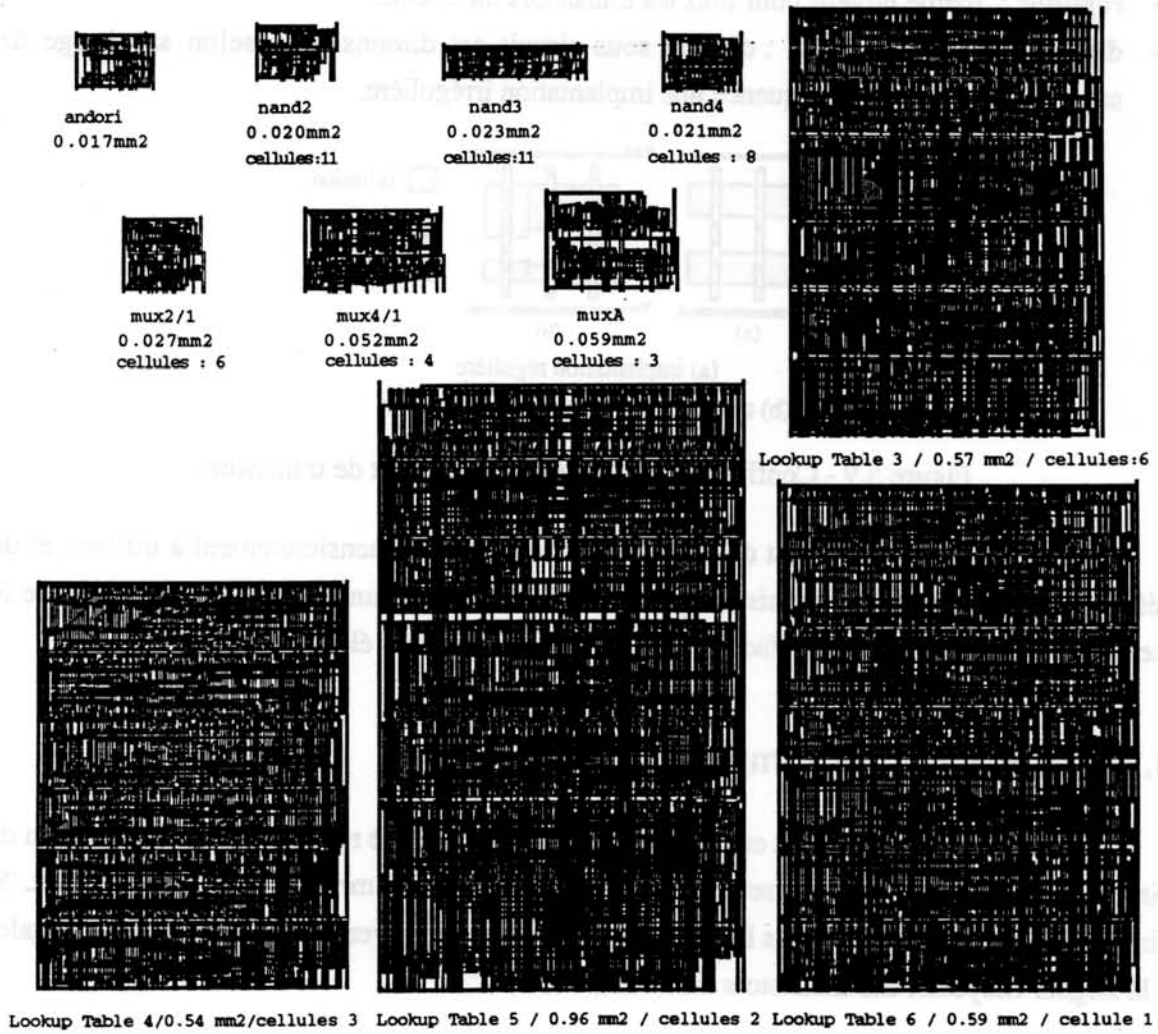


Figure 5.8 - Présentation des masques de la fonction f2 (Wn=Wp=16µm)

5.7 ANALYSE DES PERFORMANCES ELECTRIQUES

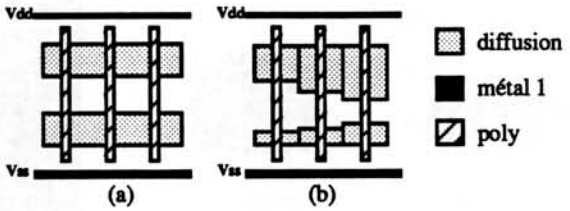
Un système de synthèse automatique des masques doit concilier deux critères distincts :

- générer les polygones constituant la représentation physique du circuit en utilisant le minimum de surface de silicium,
- obtenir les meilleures performances électriques possibles.

Généralement, la surface occupée est inversement proportionnelle aux performances, ce qui signifie que pour une surface minimale les performances sont assez faibles. Pour des bonnes performances le coût en surface est très élevé.

Pour le style d'implantation des masques *linear matrix*, deux configurations de dimensionnement de transistors sont utilisées (figure 5.9) :

- **régulier** : même largeur pour tous les transistors du circuit,
- **dimensionnement local** : chaque sous-circuit est dimensionné selon sa charge de sortie, ce qui a pour conséquence une implantation irrégulière.



(a) implantation régulière
(b) transistors dimensionnés localement

Figure 5.9 - Configurations de dimensionnement de transistors

Le but de cette étude est de conclure sur le type de dimensionnement à utiliser, et de définir les dimensions de transistors pour une technologie donnée, permettant d'atteindre le meilleur compromis entre la surface occupée et les performances électriques.

5.7.1 Protocole de simulation

Chaque entrée du circuit est contrôlée par un inverseur de référence. Si l'implantation du circuit est régulière, cet inverseur est dimensionné de la même manière que le circuit. Si l'implantation est irrégulière, les largeurs des transistors de l'inverseur de référence sont égales à la largeur moyenne des transistors du circuit.

Les sorties du circuit sont chargées par une capacité proportionnelle à la capacité d'un inverseur avec les dimensions minimales de la technologie utilisée ($C_{inv\ min}$). Cette capacité est calculée à partir de la formule :

$$C_{inv\ min} = 2 * Cox * W_{min} * L_{min} \qquad \text{(équation 5.1)}$$

Le tableau 5.5 présente les valeurs des capacités utilisées pour les 5 technologies étudiées.

	ECDM20	ECPD15	ECPD12	ECPD10	ECPD07
Cox (fF/µm²)	0,8625	1,38	1,38	1,725	2,30
W_{min} (µm)	3	2	1,5	1,25	1
L_{min} (µm)	2	1,6	1,2	1	0,8
$C_{inv\ min}$ (fF)	10,35	8,83	4,97	4,31	3,68

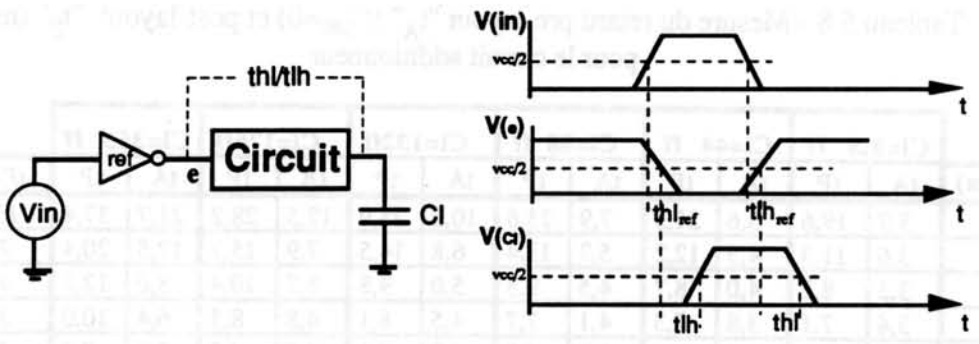
Tableau 5.5 - Valeurs de capacités de référence

Les circuits ont été simulés avec le simulateur électrique HSPICE [H90], en utilisant le niveau 2 de simulation. Le tableau 5.6 montre les paramètres de la carte de modélisation utilisée pour décrire les transistors N et P.

	ECDM20		ECPD15		ECPD12		ECPD10		ECPD07	
	N	P	N	P	N	P	N	P	N	P
Ld (µm)	0,15	0,2	0,325	0,300	0,125	0,1	0,125	0,047	0,075	0,021
Tox (A)	400	400	250	250	250	250	200	200	150	150
nsub	5,3e15	19e15	20e15	50e15	20e15	50e15	25e15	25e15	23,5e15	200e15
vto (V)	0,9	-0,9	0,7	-1,1	0,7	-1,1	0,82	-1,4	0,906	-0,917
μ_0 (cm²/vs)	510	175	510	210	510	210	690	231	553,8	220,7
u_{exp}	0,0192	0,0311	0,22	0,33	0,22	0,33	0,35	0,35	0,195	0,2168
u_{crit}	1000	4720	24300	51000	24300	51000	35000	71000	50000	17600
v_{max}	37900	37200	54000	47000	54000	47000	70800	320000	68150	70000
xj (nm)	500	600	400	500	400	500	250	250	55	550
gamma	0,49	0,92	0,65	0,87	0,65	0,87	0,76	0,78	0,807	0,618

Tableau 5.6 - Paramètres de la carte de modélisation (niveau 2, cas typique)

Les temps de montée et de descente (figure 5.10) sont mesurés entre l'entrée du circuit (V_E) et la sortie (V_{CL}), à la moitié de l'excursion de chaque signal.



thl_{ref}, thl_{ref} - temps de montée, descente de l'inverseur de référence
thl, tlh - temps de montée, descente du circuit

Figure 5.10 - Mesure du temps de montée et de descente

5.7.2 Simulations

Les tableaux ci-dessous présentent, pour la technologie ECPD15, la variation du retard en fonction des largeurs de transistors et de la charge de sortie. Les figures 5.11 et 5.12 présentent les courbes en faisant référence à ces tableaux.

W (µm)	Cl=8,8 ff	Cl=44 ff	Cl=88 ff	Cl=132 ff	Cl=176 ff	Cl=353 ff
2	0,61	1,10	1,70	2,30	2,9	5,4
4	0,36	0,61	0,92	1,23	1,53	2,76
8	0,27	0,40	0,56	0,71	0,87	1,48
12	0,24	0,33	0,43	0,54	0,64	1,06
16	0,23	0,30	0,37	0,45	0,53	0,84
20	0,22	0,27	0,33	0,40	0,46	0,71
24	0,21	0,26	0,31	0,36	0,42	0,63
28	0,21	0,25	0,30	0,34	0,39	0,57
32	0,21	0,24	0,28	0,32	0,36	0,52

Tableau 5.7 - Mesure du retard post-layout (ns) pour le circuit inverseur

	Cl=8,8 ff		Cl=44 ff		Cl=88 ff		Cl=132ff		Cl=176ff		Cl=353 ff		
W (µm)	tA	tP	tA	tP	tA	tP	tA	tP	tA	tP	tA	tP	*tP-tA*
2	1,76	6,12	2,23	6,59	2,83	7,20	3,43	7,82	4,04	8,39	6,50	10,8	*4,36*
4	1,69	3,89	1,94	4,18	2,26	4,46	2,55	4,75	2,84	5,05	4,06	6,21	*2,20*
8	1,66	3,03	1,80	3,17	1,94	3,33	2,08	3,46	2,23	3,60	2,83	4,22	*1,37*
12	1,64	2,72	1,74	2,81	1,85	2,95	1,94	3,05	2,03	3,14	2,42	3,52	*1,10*
16	1,64	2,60	1,71	2,68	1,80	2,76	1,87	2,84	1,94	2,89	2,23	3,19	*0,96*
20	1,63	2,52	1,69	2,58	1,76	2,65	1,83	2,70	1,89	2,77	2,11	2,99	*0,89*
24	1,63	2,46	1,67	2,51	1,73	2,57	1,79	2,63	1,83	2,67	2,02	2,87	*0,84*
28	1,63	2,43	1,67	2,47	1,70	2,52	1,76	2,57	1,81	2,62	1,98	2,77	*0,80*
32	1,64	2,39	1,67	2,43	1,69	2,48	1,74	2,52	1,79	2,55	1,93	2,69	*0,75*

*(tA) : simulation **sans** capacités parasites, (tP) : simulation **avec** capacités parasites,*

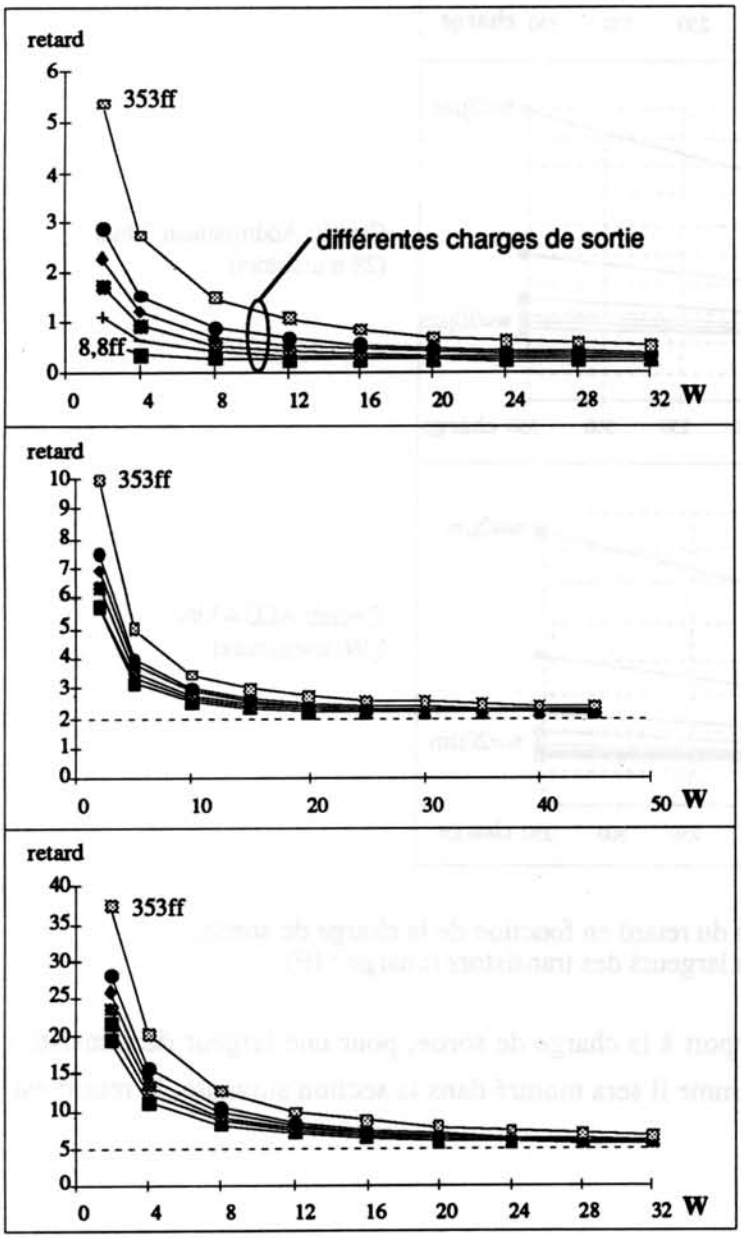
(tP-tA) : contribution des parasites au retard pour une largeur de transistor donné

Tableau 5.8 - Mesure du retard pre-layout "t_A" (C_{par}=0) et post-layout "t_p" (ns) pour le circuit additionneur

	Cl=8,8 ff		Cl=44 ff		Cl=88 ff		Cl=132ff		Cl=176ff		Cl=353 ff		
W (µm)	tA	tP	tA	tP	tA	tP	tA	tP	tA	tP	tA	tP	*tP-tA*
2	3,9	19,6	5,6	21,3	7,9	23,6	10,2	25,9	12,5	28,2	21,7	37,4	*15,7*
4	3,6	11,3	4,5	12,2	5,7	13,4	6,8	14,5	7,9	15,7	12,5	20,4	*7,7*
8	3,5	8,2	4,0	8,7	4,5	9,3	5,0	9,8	5,7	10,4	8,0	12,8	*4,7*
12	3,4	7,1	3,8	7,3	4,1	7,7	4,5	8,1	4,8	8,5	6,4	10,0	*3,6*
16	3,4	6,4	3,7	6,6	3,9	6,9	4,2	7,1	4,5	7,5	5,7	8,6	*3,0*
20	3,4	6,0	3,6	6,2	3,8	6,5	4,0	6,7	4,3	6,9	5,2	7,8	*2,6*
24	3,4	5,8	3,6	6,0	3,8	6,2	3,9	6,4	4,1	6,5	4,8	7,3	*2,4*
28	3,4	5,7	3,6	5,8	3,7	6,0	3,8	6,1	4,0	6,4	4,7	7,0	*2,3*
32	3,4	5,6	3,5	5,7	3,7	5,8	3,8	6,0	3,9	6,1	4,5	6,7	*2,2*

Tableau 5.9 - Mesure du retard pre-layout "t_A" (C_{par}=0) et post-layout "t_p" (ns) pour le circuit ALU

Notons que dans les tableaux 5.8 et 5.9 nous avons comparé le retard pre-layout et post-layout. L'écart constaté entre les simulations pre-layout et post-layout montre l'intérêt de disposer d'un générateur automatique de masques pour évaluer rapidement les performances d'une macro-cellule.



Circuit : inverseur

(2 transistors)

Circuit : Additionneur 2 bits

(28 transistors)

Circuit : ALU 4 bits

(260 transistors)

Figure 5.11 - Variation du retard en fonction des largeur de transistors (W), pour différents charges de sortie, technologie ECPD15 (simulations post-layout)

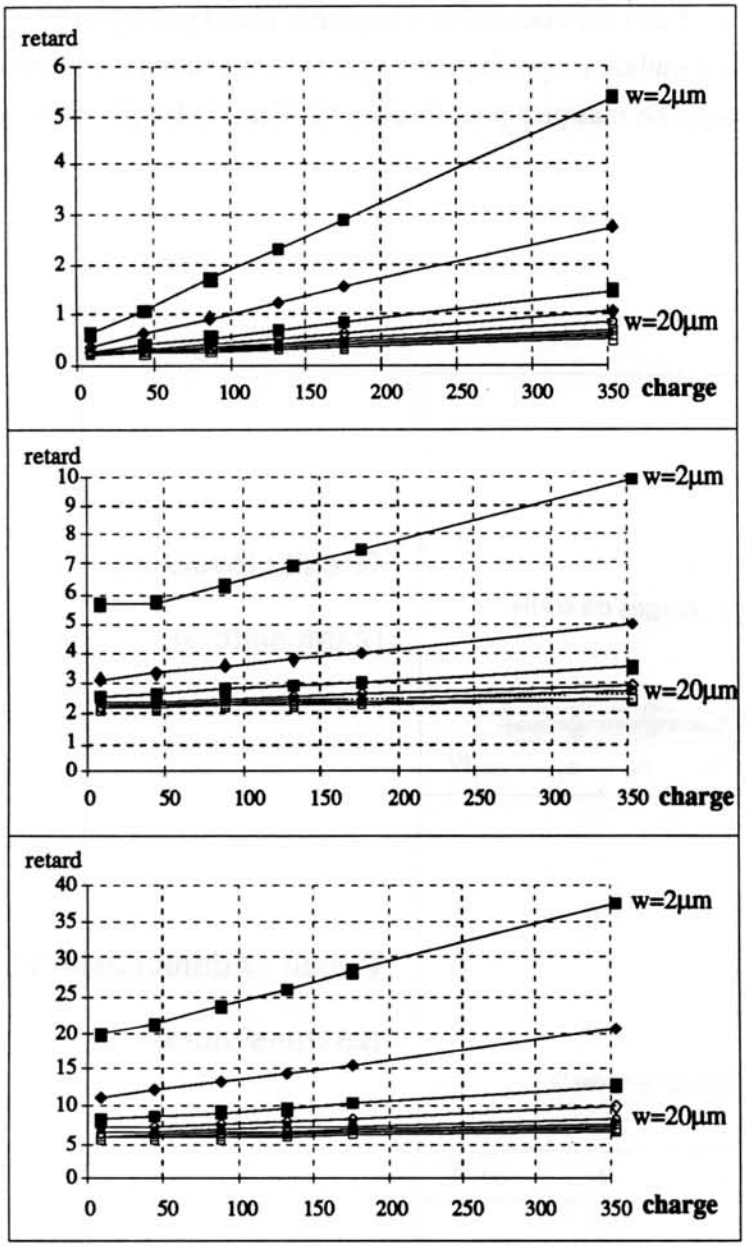


Figure 5.12 - Variation du retard en fonction de la charge de sortie, pour différentes largeurs des transistors (charge : fF)

L'évolution du retard par rapport à la charge de sortie, pour une largeur de transistor donné, est linéaire (figure 5.12). Comme il sera montré dans la section suivante, le retard est fonction de trois termes :

$$t = \alpha + \frac{\beta}{W} + \gamma \frac{C_l}{W}$$

Les termes "α" et "β" correspondent à la contribution au retard des capacités parasites des masques, nécessaires pour réaliser la cellule (par exemple, les capacités de diffusion et les capacités de routage).

Le terme "α" est la valeur asymptotique du retard (correspond à "W" infini). Le terme "β" est le parasite indépendant de la largeur des transistors. La parcelle, "β/W", représente la différence entre le retard du circuit avec parasites est sans parasites de routage.

La valeur de "W" qui minimise les termes "α" et "β" correspond au début de la région plate des courbes de la figure 5.11. La valeur de "α" est l'indicateur de l'efficacité de l'implémentation.

Ainsi, l'équation du retard d'une cellule, pour une largeur W, est donnée par trois simulations:

t_α = simulation avec W infini (100µm), à faible charge (C_l=$C_{INV\ MIN}$)

t_{SANS} = simulation sans parasites de routage, avec charge C_l

t_{PAR} = simulation avec parasites de routage, avec charge C_l

Les termes sont donc : $\alpha = t_\alpha$, $\frac{\beta}{W} = t_{PAR} - t_{SANS}$, $\gamma = \frac{W}{C_l}(t_{SANS} - t_\alpha)$

Par exemple :

- Circuit ALU, W=8µm

  t_α = 3,4 t_{PAR} (C_l=353ff) = 12,8 ns t_{SANS} (C_l=353ff) = 8 ns

$$t_{PAR} = 8{,}2 + 0{,}1 * \frac{C_l}{W} = 8{,}2 + \frac{C_l}{80}$$

- Circuit ALU, W=28µm

  t_α = 3,4 t_{PAR} (C_l=353ff) = 7 ns t_{SANS} (C_l=353ff) = 4,7 ns

$$t_{PAR} = 5{,}7 + 0{,}1 * \frac{C_l}{W} = 5{,}7 + \frac{C_l}{280}$$

- Circuit Additionneur, W=20µm

  t_α = 1,62 t_{PAR} (C_l=353ff) = 2,99 ns t_{SANS}(C_l=353ff) = 2,11 ns

$$t_{PAR} = 2{,}5 + \frac{C_l}{720}$$

L'ensemble de courbes présentées permet de démontrer qu'à partir d'une largeur de transistor donnée, le retard du circuit devient pratiquement constant et indépendant de la charge de sortie. Dans notre cas, pour la technologie ECPD15, cette valeur de largeur de grille est de 20 µm.

5.8 ANALYSE DES PERFORMANCES ÉLECTRIQUES POUR DIFFÉRENTES TECHNOLOGIES

Les courbes de la figure 5.13 illustre l'évolution du retard par rapport à la charge de sortie, pour les différentes technologies utilisées.

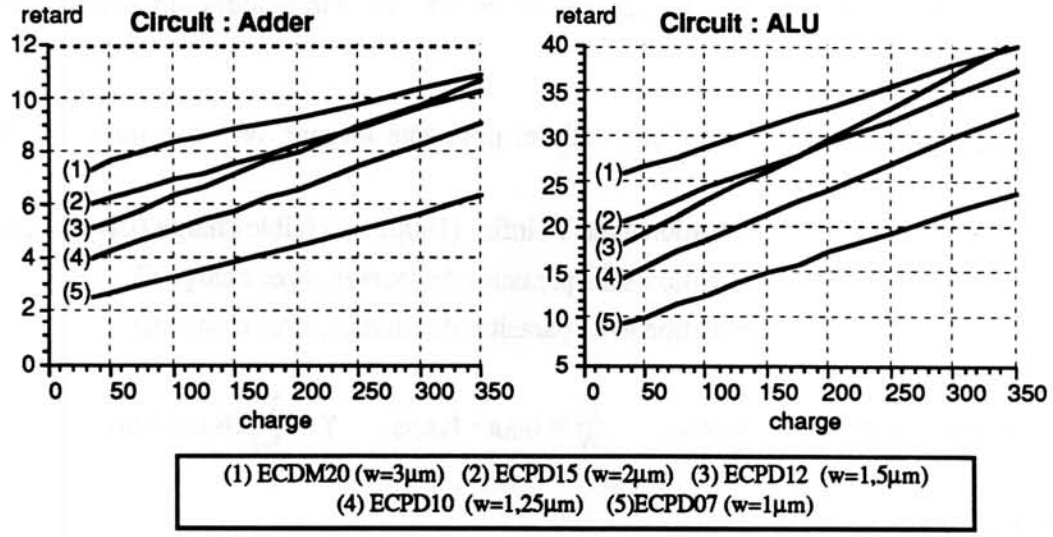


Figure 5.13 - Mesure du retard en fonction de la charge de sortie ($W=W_{min}$)

A l'aide de la formulation analytique du retard développée au LIRMM [DES88] nous pouvons écrire:

$$t_{lh,hl} = \frac{Y}{2\,X_N} * \tau_{st} \qquad \text{(équation 5.2)}$$

$$Y = \frac{C_L}{C_{ref}} = \frac{kW_N + C_{rout} + C_l}{Cox\ L_{min}\ W_{min}} \qquad \text{(équation 5.3)}$$

$$X_N = \frac{W_N}{W_{MIN}} \qquad \text{(équation 5.4)}$$

τ_{st} *représente le temps de descente d'un inverseur constitué de transistors de largeur et de longueur minimum, chargé par un inverseur identique;*

Y *représente la charge normalisée par rapport à la charge de référence (équation 5.1), en tenant compte de la capacité active d'entrée des transistors (kW_N), les capacités de routage (C_{rout}) et la charge de sortie (C_l);*

X_N *représente les largeurs des transistors N normalisées par rapport à la largeur du transistor N de référence (W_{MIN}).*

Ainsi :

$$t_{lh,hl} = \frac{kW_N + C_{rout} + C_l}{2\ Cox\ L_{min}\ W_N} * \tau_{st} \qquad \text{(équation 5.5)}$$

Le retard est donc fonction de trois termes :

$$t_{lh,hl} \approx \alpha + \frac{\beta}{W} + \gamma\ \frac{C_l}{W} \qquad \text{(équation 5.6)}$$

- le retard dû à la charge intrinsèque (α),
- le rapport entre la capacité de routage (parasite) et la largeur des transistors ($\frac{\beta}{W}$),
- le rapport entre la capacité de sortie et la largeur des transistors ($\gamma \frac{C_l}{W}$).

- L'utilisation de la largeur minimale (W_{MIN}) pour la génération des macro-cellules est déconseillée, car l'augmentation du retard en fonction de la charge de sortie est très importante (*facteur d'extrapolation*).

- L'utilisation des largeurs de transistors plus importantes permet d'obtenir des retards sensiblement constants, indépendamment de la variation de la charge ($w \to \infty \Rightarrow t_{lh,hl} \to \alpha$).

La figure 5.14 illustre les courbes du retard en fonction des largeurs de transistors, pour les technologies 1.2µm, 1.0µm et 0.7µm.

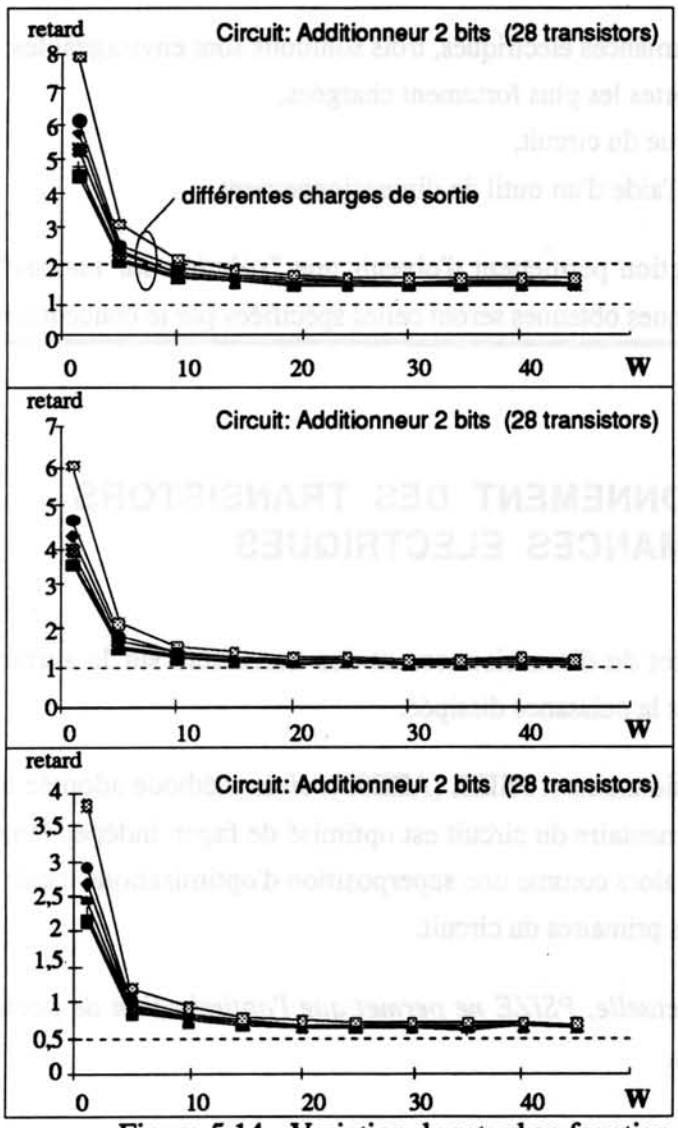


Figure 5.14 - Variation du retard en fonction des largeur de transistors, pour différents charges de sortie et technologies

L'ensemble de courbes présentées montre qu'à partir d'une largeur de transistor donné, le retard devient sensiblement constant en fonction de la charge de sortie. Nous remarquons que le début de cette région stable correspond à des valeurs de largeur égale à **10** foix la largeur minimale d'une technologie donnée (20 µm pour ECPD15, 15µm pour ECPD12, 12.5 µm pour ECPD10 et 10µm pour ECPD07).

Le fait d'augmenter les largeurs des transistors après la zone de stabilité n'apporte pas d'amélioration importante sur les performances électriques. Ceci implique par contre une augmentation importante de la puissance dissipée et de la surface occupée.

L'implantation régulière, avec des largeurs de transistors ayant une dimension égale à 10 fois la largeur minimale de la technologie, est une solution qui permet d'atteindre le meilleur compromis temps-surface. Le problème de cette approche régulière est la dissymétrie entre le temps de monté et de descente.

Afin d'avoir de meilleures performances électriques, trois solutions sont envisageables:
- dimensionner seulement les portes les plus fortement chargées,
- dimensionner les étages de sortie du circuit,
- dimensionner les transistors à l'aide d'un outil de dimensionnement.

Ces trois techniques d'optimisation permettent d'obtenir une "*solution sur mesure*", c'est à dire que les performances électriques obtenues seront celles spécifiées par le concepteur.

5.9 EFFET DU DIMENSIONNEMENT DES TRANSISTORS DANS LES PERFORMANCES ELECTRIQUES

Nous étudions maintenant l'effet du dimensionnement des transistors sur la surface occupée, les performances électriques et la puissance dissipée.

Nous utilisons l'outil de dimensionnement PSIZE [AZE92], où la méthode adoptée est l'*optimisation locale*. Chaque bloc élémentaire du circuit est optimisé de façon indépendante. L'optimisation d'un circuit se présente alors comme une superposition d'optimisations locales, en remontant des sorties vers les entrées primaires du circuit.

Remarque : dans sa version actuelle, PSIZE ne permet que l'optimisation de portes élémentaires : nands, nors et inverseurs.

Le premier exemple, l'additionneur implanté à l'aide de portes élémentaires, contient 40 transistors. Nous comparons la solution dimensionnée à la solution régulière. Les temps critiques dans les 2 approches doivent être égaux. Les résultats de cette comparaison sont présentés dans le tableau 5.10. Les cinq paramètres évalués sont :

- la dissymétrie entre le temps de montée et de descente ($| t_M - t_D |$),
- la puissance moyenne dissipée (mesurée par HSPICE, en utilisant la commande de mesure de la puissance moyenne, voir la figure 2.6 - simulation HSPICE),
- la surface active (ΣW),
- le total des capacités parasites (ΣC_{PAR}),
- la surface de la macro-cellule générée.

Circuit : additionneur Technologie : ECPD15 Cl = 706 fF	Solution régulière W = 20 µm	Solution dimensionnée	Solution dimensionnée / Solution régulière
t_M - temps de montée (ns)	**4,0**	3,9	
t_D - temps de descente (ns)	3,3	**4,1**	
$\| t_M - t_D \|$ (ns)	0,7	0,2	
Puissance (mW)	1,4	1,3	0,93
ΣW (µm)	800	620	0,78
ΣC_{PAR} (pF))	1,06	1,02	
Surface (mm^2)	0,0185	0,0210	**1,13**

Tableau 5.10 - Comparaison entre la solution dimensionnée et la solution régulière, pour le circuit additionneur

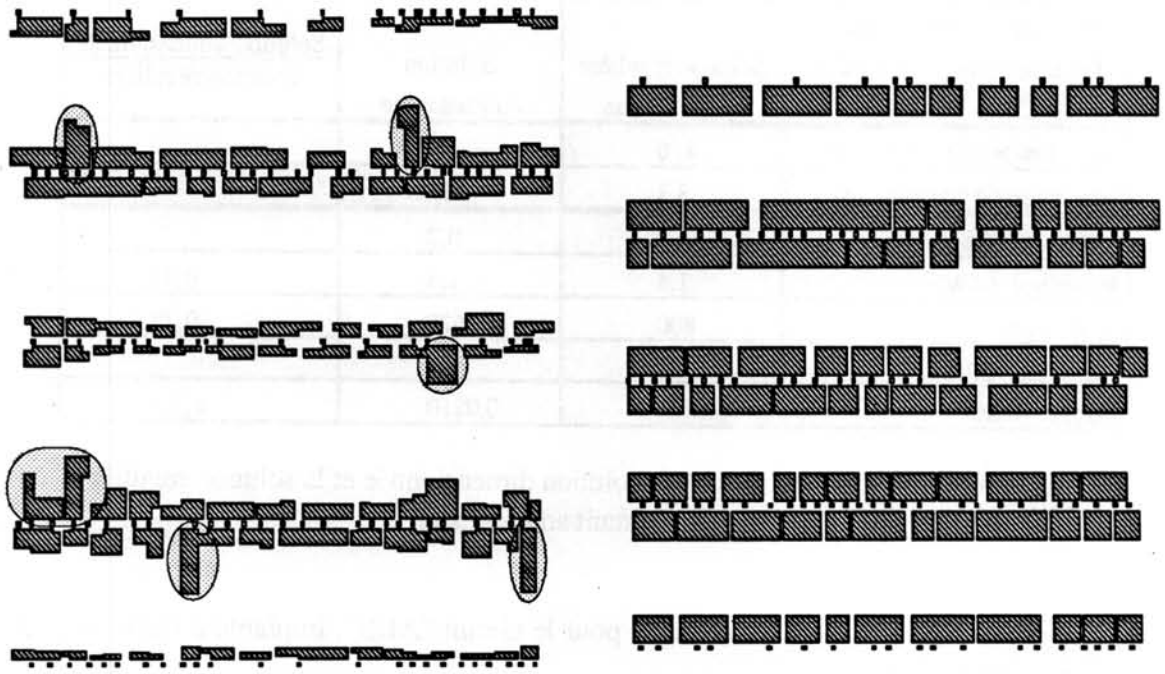
Le tableau 5.11 montre les résultats pour le circuit "ALU", implanté à l'aide de portes élémentaires (432 transistors).

Circuit : "ALU" Technologie : ECPD15 Cl = 706 fF	Solution régulière W = 27 µm	Solution dimensionnée	Solution dimensionnée / Solution régulière
t_M - temps de montée (ns)	**10,8**	**10,9**	
t_D - temps de descente (ns)	5,7	8,4	
$\| t_M - t_D \|$ (ns)	5,1	2,5	
Puissance (mW)	11,3	9,9	0,88
ΣW (µm)	11664	6563	0,56
ΣC_{PAR} (pF))	20,9	20,5	
Surface (mm^2)	0,336	0,397	**1,18**

Tableau 5.11 - Comparaison entre la solution dimensionnée et la solution régulière, pour le circuit "ALU"

La comparaison entre la solution régulière (27 µm) et la solution dimensionnée (circuit ALU) montre une diminution non proportionnelle entre la puissance dissipée (0,87) et la surface active total (0,56). Ceci est dû au fait que la largeur des transistors dans les chemins sensibilisés par la simulation (chemins critiques) sont équivalents dans les deux approches. Ce résultat montre que la puissance dissipée dans une structure est liée non seulement à la surface active, mais aussi à l'activité des portes.

Nous remarquons aussi que le dimensionnement des transistors permet d'obtenir des temps de propagation plus symétriques. Cependant, le style d'implantation utilisé (linear matrix), impose une surface occupée plus importante pour la solution dimensionnée que pour la solution régulière (figure 5.15).



(a) Solution dimensionnée (b) Solution régulière

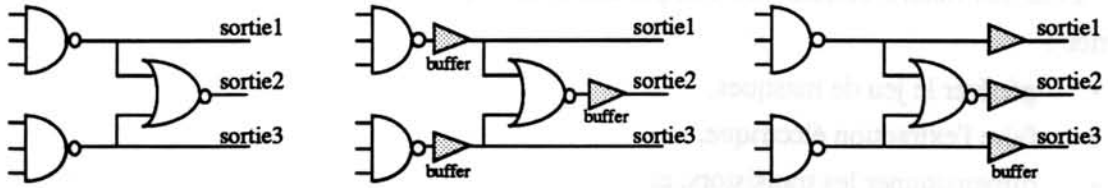
Figure 5.15 - Niveau diffusion pour les solutions dimensionnée et réguliere

La figure 5.15 montre qui l'augmentation de surface est due à quelques transistors ayant une largeur supérieure à la moyenne des transistors du circuit. Ces transistors font partie des cellules fortement chargées (internes au circuit) et aux cellules de sortie. Deux solutions pour éviter cette perte en surface sont envisageables : parallèlisation des transistors les plus larges et insertion d'étages tampons.

La première solution, parallèlisation des transistors, n'est réalisable que sur les inverseurs. La parallèlisation d'autres cellules plus complexes (*nands, nors, AOIs*) rend le routage interne à la cellule plus complexe et double le nombre des transistors, ce qui implique une importante perte en surface.

La deuxième solution, insertion d'étages tampons (*buffers*), permet de réduire la hauteur de la cellule ayant les transistors les plus larges et d'avoir une réponse plus constante en fonction de la charge de sortie (voir par exemple, dans la figure 5.12, la courbe relative au inverseur).

Remarque : le "buffer" est pour nous un inverseur constitué des transistors de largeur W, suivi d'un inverseur avec largeur 2W. Le deuxième inverseur est implanté à l'aide de 4 transistors, 2 en parallèle dans le plan P est deux en parallèle dans le plan N. Ceci permet de respecter la contrainte de la solution régulière (tous les transistors ayant la même largeur).



(a) Circuit Original

(b) Insertion des buffers sans isolation des noeuds internes

(c) Insertion des buffers **avec** isolation des noeuds internes

Figure 5.16 - Insertion des buffers dans les cellules de sortie

La meilleure façon d'insérer les buffers dans les sorties du circuit est présentée dans la figure 5.16.c. L'avantage de cette topologie par rapport à celle de la figure 5.16.b est l'isolation des noeuds internes du circuit. Ainsi, la variation de la charge n'est appliquée que sur les *buffers*, et n'a pas d'effet sur les noeuds internes du circuit.

Le tableau ci-dessous présente les résultats de la comparaison entre les topologies initiales et avec des buffers dans les sorties. Les meilleures performances (temps et surface) sont obtenues avec la solution régulière, contenant des buffers dans les sorties. Autre avantage des solutions avec buffers est la faible sensibilité à la variation de la charge de sortie, dans l'exemple, 0.4 ns.

Circuit : ALU	Topologie initiale		buffers dans les sorties	
Tech : ECPD15	Régulière w=27µm	Dimensionnée	Régulière w=15µm	Dimensionnée
Transistors #	432	432	470	470
t_M/t_D Cl=706 ff	10,8 / 5,7	10,9 / 8,4	10,0 / 6,6	10,5 / 8,6
t_M/t_D Cl=353 ff	9,4 / 5,4	10,2 / 7,9	9,7 / 6,4	10,3 / 8,4
t_M/t_D Cl=176 ff	8,7 / 5,2	9,9 / 7,6	9,6 / 6,3	10,1 /8,2
Pm Cl=706 ff	11,3	9,95	8,9	9,5
Pm Cl=353 ff	10,4	9,90	8,0	8,6
Pm Cl=176 ff	10,0	8,50	7,6	8,2
X (µm) *x* Y(µm)	602 x 558	618 x 642	602 x 457	640 x 574
Surface (mm^2)	0,3359	0,3968	0,2833	0,3674

t_M/t_D - temps de montée et de descente (ns) ♦ *Pm - puissance moyenne (mW)*

Tableau 5.12 - Insertion de buffers dans les sorties du circuit

La solution dimensionnée a permis le calcul de la largeur moyenne des transistors:

$$W \text{ moyenne} = \frac{\Sigma W_N + \Sigma W_P}{\text{nombre total de transistors}}$$

Pour la génération de la solution régulière nous pouvons utiliser la largeur moyenne obtenue à partir de la solution dimensionnée. La solution régulière, obtenue à partir d'une largeur égale à 10 fois la valeur minimale (w=20 µm pour la technologie ECPD15) peut également être utilisée, car cette valeur est très proche de la solution dimensionnée.

Pour les macro-cellules de complexité plus importante (1000 à 2000 transistors) la séquence :

- générer le jeu de masques,
- faire l'extraction électrique,
- dimensionner les transistors, et
- générer les masques de la solution optimisée

est très coûteuse en temps (l'étape critique étant le compactage de masques).

Ainsi, pour que le concepteur puisse réellement utiliser l'outil de dimensionnement, il est nécessaire d'ajouter au système de synthèse de masques deux modules :

- un module de prédiction de parasites, qui prend en compte le style de layout, afin d'éviter les 2 générations de masques,
- l'optimisation de portes complexes et de portes de transmission.

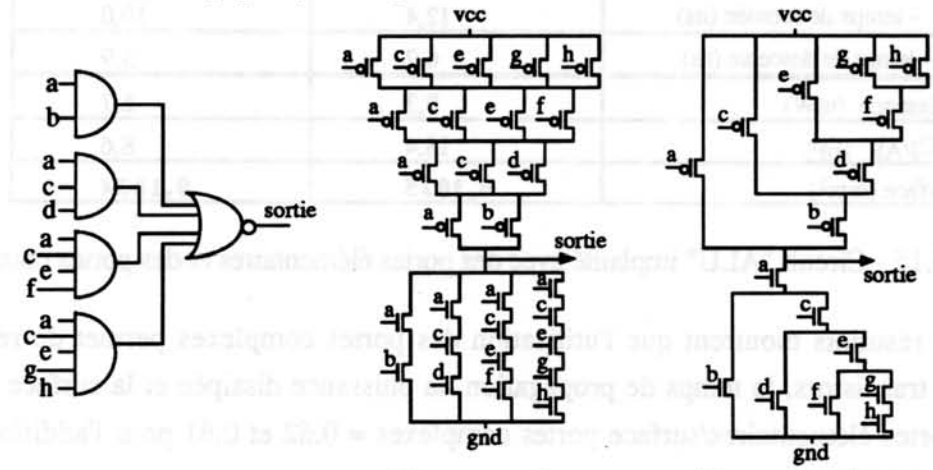
Cette étude a montré que la solution dimensionnée implique une augmentation de surface, due au style d'implantation des masques. Dans l'approche *linear matrix*, l'implantation régulière est la solution qui apporte le meilleur compromis temps-surface. La technique conseillée pour accélérer le circuit, si cela est nécessaire, est donc l'insertion d'étages tampons après les cellules fortement chargées et sur les noeuds de sortie du circuit, ce qui les isole de la charge de sortie.

5.10 IMPACT DE L'UTILISATION DE PORTES SIMPLES ET DE PORTES COMPLEXES

Pour mettre en évidence les avantages de l'utilisation des portes complexes sur les portes élémentaires (*nands*, *nors*, *inverseurs*), nous comparons les performances électriques et de surface occupée des circuits implantés à l'aide de ces portes.

L'implantation d'une fonction logique avec des portes complexes permet de réduire le nombre de transistors et de couches logiques du circuit. L'exemple de la figure 5.17 illustre un cas typique de circuit numérique, portes *and* suivies par une porte *nor*. La porte de la figure 5.17.b est obtenue directement à partir du schéma logique. Chaque porte *and* est une branche série, et la porte *nor* est constituée par la liaison parallèle des ces branches (pour le plan N).

Si les entrées de la porte complexe sont factorisées (suppression des termes redondantes), nous obtenons la porte de la figure 5.17.c. Le tableau 5.13 présente le nombre de transistors et de couches logiques pour implanter cette fonction.



(a) schéma logique

(b) porte complexe, sans factorisation

(c) porte complexe, factorisée

Figure 5.17 - Schéma électrique d'une porte complexe

Implantation de la porte complexe de la figure 5.17 :	Nombre de transistors	Nombre de couches logiques
portes élémentaires	38	3
AOIs sans factorisation	28	1
AOIs factorisées	16	1

Tableau 5.13 - Nombre de transistors et de couches logiques pour différentes implantations de la cellule de la figure 5.17

Le tableau 5.14 présente les résultats de la comparaison entre l'implémentation du circuit additionneur généré avec des portes élémentaires et des portes complexes (factorisées). Le tableau 5.15 présente les résultats pour le circuit "ALU".

Circuit : additionneur Technologie : ECPD15 w=20µm- Cl=706fF	Portes élémentaires	Portes complexes
Nombre de transistors	**40**	**28**
Nombre de portes	10	4
t_M - temps de montée (ns)	4,0	3,3
t_D - temps de descente (ns)	3,3	2,0
Puissance (mW)	1,4	1,35
ΣC_{PAR} (pF)	1,06	0,5
Surface (mm^2)	**0,0185**	**0,0115**

Tableau 5.14 - Circuit additionneur implanté avec des portes élémentaires et des portes complexes

Circuit : ALU Technologie : ECPD15 w=20µm- Cl=706fF	Portes élémentaires	Portes complexes
Nombre de transistors	**432**	**262**
Nombre de portes	101	39
t_M - temps de montée (ns)	12,4	10,0
t_D - temps de descente (ns)	6,2	5,9
Puissance (mW)	9,3	6,7
ΣC_{PAR} (pF)	18,4	8,6
Surface (mm^2)	**0,3025**	**0,1834**

Tableau 5.15 - Circuit "ALU" implanté avec des portes élémentaires et des portes complexes

Ces résultats montrent que l'utilisation des portes complexes permet de réduire le nombre de transistors, le temps de propagation, la puissance dissipée et la surface occupée (surface portes élémentaires/surface portes complexes = 0,62 et 0,61 pour l'additionneur et l'ALU respectivement).

Remarque : il faut respecter le nombre de transistors en série (conseillé : cinq pour le plan N et quatre pour le plan P).

La description au niveau des portes élémentaires est soit créée par le concepteur, soit générée automatiquement à partir d'un outil de synthèse haut niveau.

La conversion d'une liste de portes élémentaires sur une bibliothèque de fonctions est appelée assignement technologique ("*mapping*"). La qualité du "*mapping*" sur une bibliothèque de cellules pré-caractérisées (standard-cell, par exemple) est fonction du nombre de portes disponibles. Comme le nombre de portes complexes différentes réalisables est très élevé (voir tableau 3.1), le "*mapping*" sur une bibliothèque de cellules est assez limité.

Étant donnée la flexibilité de notre générateur, la réalisation d'un module informatique qui réalise le "*mapping*" indépendamment de bibliothèques de fonctions permettra d'obtenir une solution optimisée en surface et en temps de propagation. La factorisation des AOIs est aussi nécessaire, car elle permet de réduire le nombre de transistors, et par conséquence la surface occupée.

5.11 COMPARAISON AVEC L'APPROCHE STANDARD-CELL

Nous comparons maintenant les résultats de la surface occupée et des performances électriques (tableaux 5.16 et 5.17), pour les circuits générés à l'aide du générateur TROPIC et de l'outil standard-cell de CADENCE™, en utilisant une bibliothèque de cellules pré-caractérisés industrielle. Les circuits choisis sont l'ALU et un compteur synchrone (LS161).

Circuit : ALU - (AOIs) Technologie : ECPD15	W = 3 µm	W = 16 µm	W = 30 µm	Standard-Cells (Cl=200fF)
X(µm) - Y(µm)	390 - 325	373 - 398	385 - 466	730 - 411
Surface (mm^2)	0,127	0,148	0,179	0,30
$t_{Montée}/t_{Descente}$ (Cl=353 ff)	27,3 / 14,6	9,3 / 5,9	7,2 / 4,8	
$t_{Montée}/t_{Descente}$ (Cl=706 ff)	32,2 / 15,7	11,3 / 6,4	8,3 / 5,1	
$t_{Montée}/t_{Descente}$ MAX				21,7 / 17,1
$t_{Montée}/t_{Descente}$ TYP				11,1 / 8,5
$t_{Montée}/t_{Descente}$ MIN				4,5 / 3,7

(a) Synthèse du circuit ALU en utilisant des portes complexes (260 transistors)

Circuit : ALU - (portes) Technologie : ECPD15	W = 3 µm	W = 16 µm	W = 30 µm	Standard-Cells (Cl=200fF)
X(µm) - Y(µm)	612 - 381	612 - 478	612 - 584	730 - 411
Surface (mm^2)	0,2332	0,2925	0,3574	0,30
$t_{Montée}/t_{Descente}$ (Cl=353 ff)	34,0 / 14,0	11,5 / 6,1	9,2 / 5,1	
$t_{Montée}/t_{Descente}$ (Cl=706 ff)	46,6 / 16,9	13,9 / 6,6	10,4 / 5,4	

(b) Synthèse du circuit ALU en utilisant des portes simples (432 transistors)

Tableau 5.16 - Implantation du circuit ALU à l'aide de TROPIC et d'un outil "*standard-cells*"

Circuit : LS161 Technologie : ECPD15	W = 3 µm	W = 16 µm	W = 30 µm	Standard-Cells (Cl=200fF)
X(µm) - Y(µm)	339 - 297	329 - 380	347 - 458	466 - 368
Surface (mm^2)	0,1007	0,1250	0,1590	0,170
$t_{Montée}/t_{Descente}$ (Cl=353 ff)	16,6 / 5,6	6,2 / 2,4	5,0 / 2,0	
$t_{Montée}/t_{Descente}$ (Cl=706 ff)	19,9 / 6,6	6,8 / 2,6	5,4 / 2,1	
$t_{Montée}/t_{Descente}$ MAX				13,1 / 13,1
$t_{Montée}/t_{Descente}$ TYP				6,8 / 6,8
$t_{Montée}/t_{Descente}$ MIN				2,7 / 2,7

Tableau 5.17 - Implantation du circuit LS161 à l'aide de TROPIC et d'un outil standard-cells

La comparaison n'est pas rigoureuse, car la taille des transistors et la façon de simuler les circuits sont différentes. Dans notre générateur les circuits sont générés avec tous les transistors ayant la même largeur (W=3µm, 16µm et 30 µm), et ils sont simulés électriquement (simulateur HSPICE, niveau 2 typique), en tenant compte de toutes les capacités parasites. Dans l'approche standard-cell les cellules de base sont dessinées manuellement, et le circuit est simulé au niveau logique (simulateur SILOS). Toutefois le simulateur logique - temporel SILOS est étalonné par le fournisseur de la bibliothèque avec les paramètres de la technologie.

Si nous considérons la surface active total (SA) comme paramètre de puissance, nous avons pour le circuit ALU :

- Standard Cell : **SA = 5258** (somme des largeurs des transistors pour chaque porte)
- TROPIC (implantation avec portes simples): **SA = 5184** (W_N=W_P=12µm, 432 trans.)
- TROPIC (implantation avec portes complexes): **SA = 3120** (W_N=W_P=12µm, 260 trans.)

La largeur choisie pour la solution régulière, 12 µm, est égale à 10 fois la largeur minimale de la technologie (ECPD10), et correspond aussi à la largeur moyenne des transistors de la bibliothèque "*standard-cell*". Les données de la surface active totale montrent que la méthode de synthèse automatique de masques peut apporter une réduction importante de la puissance dissipée, car le circuit peut être dimensionné selon les contraintes imposées par le concepteur, ou la topologie améliorée par l'utilisation de portes complexes.

Résumons les principaux avantages de l'utilisation d'un outil de synthèse automatique de masques par rapport à l'approche "*standard-cells*" :

- Le retard et la puissance dissipée du circuit sont modulables, en fonction des largeurs des transistors.
- La migration technologique est simple, il suffit de changer le fichier contenant les règles de dessin.
- La surface occupée est moins importante.
- Il n'y a pas de contraintes lors du "*mapping*", car les bibliothèques de cellules pré-caractérisées ne sont pas utilisées.

5.12 COMPARAISON AVEC L'OUTIL "LAS"

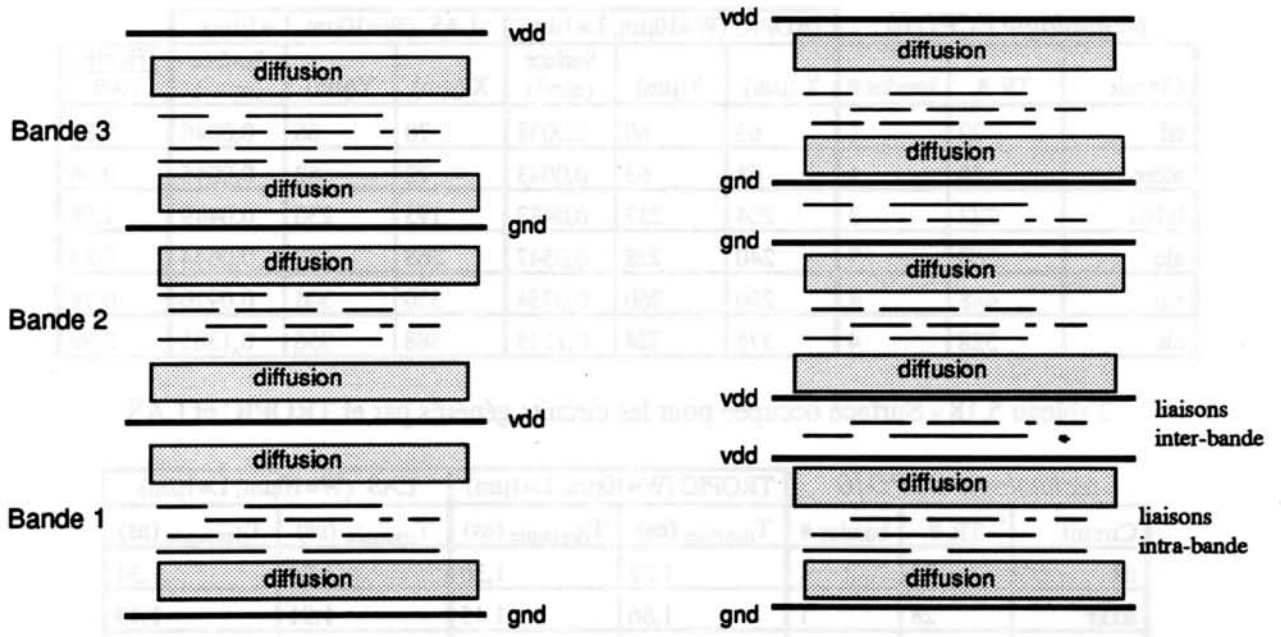
Nous présentons maintenant la comparaison entre notre approche et le générateur industriel LAS ("VIRTUOSO LAYOUT SYNTHESIZER"), lequel est intégré dans l'environnement OPUS (CADENCE™).

Les caractéristiques communes entre les générateurs TROPIC et LAS sont :

- compatibilité du fichier de la description d'entrée (syntaxe SPICE),
- même style d'implantation (linear-matrix multi-bandes),
- largeur des transistors est variable,
- absence de bibliothèques pré-caractérisées.

Les principales différences sont:

- Dans TROPIC toutes les connexions sont réalisées entre les transistors. Dans LAS les connexions internes à une bande sont réalisées entre les transistors, et les liaisons entre les bandes sont réalisées entre les bandes (figure 5.18).



(a) Topologie TROPIC : toutes les liaisons sont réalisées entre les lignes de diffusion

(b) Topologie LAS : les liaisons internes aux bandes sont réalisées entre les lignes de diffusion, et les liaisons entre les bandes sont réalisés entre les bandes

Figure 5.18 - Topologie de routage pour TROPIC et LAS

- LAS permet des liaisons verticales en polysilicium entre les bandes. Ce type de connexion augmente les résistances parasites, et par conséquent les performances électriques peuvent être dégradées.

- Les broches d'entrée/sortie, dans TROPIC, sont placées sur la périphérie de la macro-cellule, en métal 2. Le seul degré de liberté est le choix du bord (nord, sud, est, ouest). Dans LAS, il est possible de choisir le(s) bord(s) (le même signal peut être placé sur un ou plusieurs bords), l'ordre du signal par rapport aux autres signaux et le niveau (métal 1, métal 2 ou polysilicium).

- Le générateur TROPIC ne génère que des circuits en logique CMOS complémentaire et avec portes de transmission. Avec LAS il est possible de générer aussi des circuits en logique dynamique, où le nombre de transistors N et P est différent.

- Le générateur TROPIC insère le nom des signaux sur le routage. Ceci est important lors de l'extraction électrique, car le fichier résultant de l'extraction contiendra les mêmes noms des signaux que le fichier de départ. Ainsi, il est possible de repérer facilement les capacités parasites sur un signal interne au circuit, et dimensionner un chemin donné.

Les tableaux suivants (5.18 et 5.19) présentent la surface occupée et les performances électriques pour les circuits générés avec TROPIC et LAS.

technologie ECPD10			TROPIC (W=10µm, L=1µm)			LAS (W=10µm, L=1µm)			
Circuit	TR #	bandes #	X (µm)	Y(µm)	Surface (mm^2)	X (µm)	Y(µm)	Surface (mm^2)	TROPIC / LAS
dff	22	1	63	60	0,0038	70	66	0,0046	0,83
adder	28	1	68	63	0,0043	72	62	0,0045	0,96
ls161	222	3	204	237	0,0483	195	230	0,0449	1,08
alu	260	3	240	228	0,0547	265	243	0,0644	0,85
rip	448	4	290	260	0,0754	320	304	0,0970	0,78
cla	528	4	375	324	0,1215	388	356	0,1381	0,88

Tableau 5.18 - Surface occupée pour les circuits générés par et TROPIC et LAS

technologie ECPD10			TROPIC (W=10µm, L=1µm)		LAS (W=10µm, L=1µm)	
Circuit	TR #	bandes #	$T_{montée}$ (ns)	$T_{descente}$ (ns)	$T_{montée}$ (ns)	$T_{descente}$ (ns)
dff	22	1	1,82	1,30	1,85	1,34
adder	28	1	1,86	1,45	1,91	1,49
ls161	222	3	3,47	1,20	2,92	1,08
alu	260	3	5,43	3,26	5,34	3,04
rip	448	4	44,80	43,23	45,85	43,80
cla	528	4	11,27	8,10	10,24	7,25

Tableau 5.19 - Performances électriques pour les circuits générés par et TROPIC et LAS

La figure 5.19 illustre les masques du circuit ALU-4 bits.

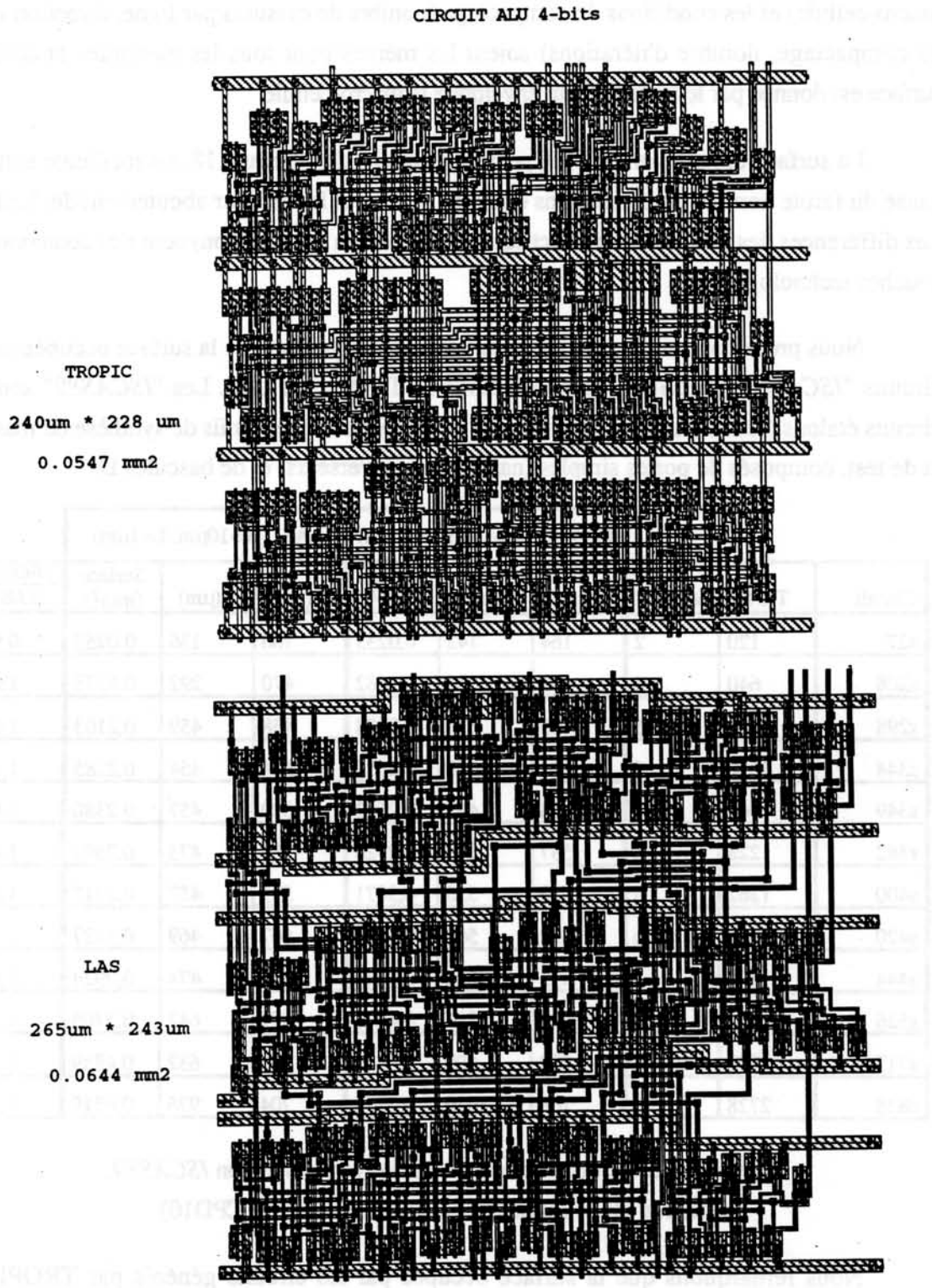


Figure 5.19 - Masque du circuit ALU-4 bits

Pour la génération des ces exemples nous utilisons une procédure écrite en *skill* (langage de programmation de CADENCE™). Ainsi nous garantissons que les conditions de génération (largeur des alimentations, nombre de bandes, position des entrées/sorties sur la périphérie de la macro-cellule) et les conditions de compactage (nombre de cassures par ligne, direction initial de compactage, nombre d'itérations) soient les mêmes pour tous les exemples étudiés. La surface est donnée par le rectangle qui enveloppe la macro-cellule.

La surface utilisée par TROPIC, dans les circuits du tableau 5.18, est meilleure surtout à cause du faible nombre de connexions entre les bandes et le meilleur aboutement des cellules. Les différences des performances électriques sont dues au routage (longueur des connexions et couches technologiques).

Nous présentons ci-dessous (tableau 5.20) la comparaison de la surface occupée par les circuits "*ISCAS89*" [BRG89], générés à l'aide de TROPIC et LAS. Les "*ISCAS89*" sont des circuits étalons ("*benchmarks*"), utilisés pour la comparaison des outils de synthèse de masques et de test, composés de portes simples (nands, nors, inverseurs) et de bascules D.

			TROPIC (W=10µm, L=1µm)			LAS (W=10µm, L=1µm)			
Circuit	TR #	bandes #	X (µm)	Y(µm)	Surface (mm^2)	X (µm)	Y(µm)	Surface (mm^2)	TROPIC / LAS
s27	120	2	164	142	0,0233	186	136	0,0253	0,92
s208	640	4	442	335	0,1482	470	292	0,1373	1,08
s298	946	6	426	537	0,2288	458	459	0,2103	1,09
s344	1034	6	505	498	0,2518	504	454	0,2285	1,10
s349	1044	6	538	483	0,2602	520	457	0,2380	1,09
s382	1228	6	537	545	0,2926	569	473	0,2692	1,09
s400	1262	6	559	550	0,3071	595	457	0,2717	1,13
s420	1336	6	618	560	0,3464	667	469	0,3127	1,11
s444	1304	7	495	615	0,3044	617	474	0,2924	1,04
s526	1604	7	599	714	0,4274	577	642	0,3705	1,15
s713	1898	7	789	690	0,5446	765	632	0,4839	1,13
s838	2728	10	768	1010	0,7765	704	926	0,6519	1,19

Tableau 5.20 - Surface occupée par les circuits étalon *ISCAS89*, générés par et TROPIC et LAS (technologie ECPD10)

Nous remarquons que la surface occupée par les circuits générés par TROPIC et d'environ 12% plus grande que la surface occupée par LAS. Ceci est dû :

- au plus grand nombre de connexions entre les bandes,
- à l'effet d'"échelle" dans le compactage.

La figure 5.20 illustre les masques du circuit s208.

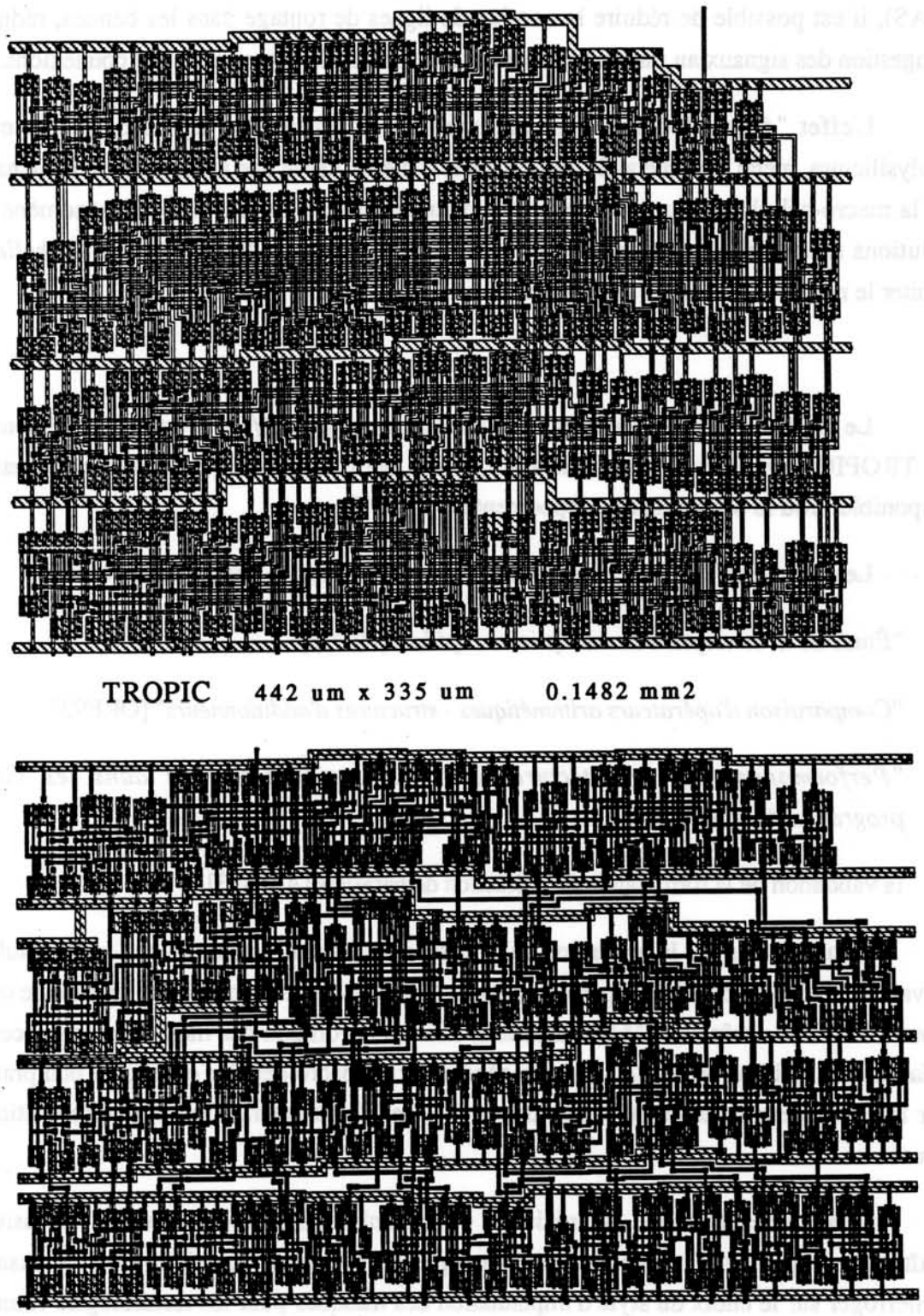


Figure 5.20 - Masque du circuit s208

La réalisation de la connexion des signaux inter-bandes entre les lignes de diffusion implique une augmentation des lignes de routage nécessaires, et par conséquent une hauteur de bande plus importante. Quand les connexions inter-bandes sont réalisées entre les bandes (LAS), il est possible de réduire le nombre de lignes de routage dans les bandes, réduire la congestion des signaux au milieu de la bande, et de réduire la longueur de ces connexions.

L'effet "*échelle*" dans le compactage survient quand des lignes de même niveau (polysilicium, métal 1 ou métal 2) sont proches. Cet effet cause une augmentation de la hauteur de la macro-cellule, et par conséquent une perte en surface. Pour éviter ce phénomène deux solutions sont envisageables : éloigner les lignes susceptibles de générer une "*échelle*", ou limiter le nombre de cassures dans les fils.

Le générateur LAS a été installé au laboratoire en janvier/1994, après l'implémentation de TROPIC. Avant l'installation de LAS, il n'y avait pas d'outil de synthèse de masques disponible, d'où le besoin du développement de TROPIC.

Le générateur TROPIC a permis d'autre part la réalisation des études suivantes :

- "*Étude de la conception de multiplieurs rapides*" [NID93]
- "*Comparaison d'opérateurs arithmétiques - structures d'additionneurs*" [ORE93]
- "*Performances des architectures de blocs logiques utilisés dans les circuits programmables*" [ROB94]
- la validation de la formulation de prédiction de parasites [AMA93]

Nous n'avons pas la prétention de rivaliser avec un outil industriel, qui est le résultat du travail d'une équipe de développement. Notons toutefois que la disponibilité récente de ce type d'outil dans une chaîne de CAO industrielle montre l'évolution des méthodes de conception. Néanmoins, l'utilisation de LAS comme générateur de macro-cellules est encore peu pratiquée par le concepteur système en raison des problèmes de caractérisation et d'optimisation des structures.

Dans notre démarche, la prédiction et l'optimisation des performances nécessitent la maîtrise complète de la synthèse automatique de masques. Aussi, il est intéressant de s'interroger sur le choix du style d'implantation des masques pour les technologies futures : 3 niveaux de métallisation et contacts empilés. Ces raisons justifient la poursuite des investigations dans ce domaine de recherche.

L'avantage de TROPIC réside dans la simplicité d'utilisation et dans la possibilité d'intégration avec le module de dimensionnement des transistors. La différence de 12% en plus de surface occupée par rapport à LAS est due surtout au choix du routage vertical. Ainsi, nous pouvons conclure que TROPIC est un outil performant qui peut être utilisé par d'autres groupes de conception de CIs ayant besoin de synthèse automatique des masques pour la génération des circuits, la validation d'architectures ou la caractérisation de macro-blocs fonctionnels.

Nous avons étudié dans ce chapitre les paramètres contrôlant la surface occupée et les performances électriques des circuits générés dans le style linear-matrix multi-bandes. Nous avons aussi comparé notre générateur à l'approche standard-cells et au générateur LAS.

Les conclusions les plus importantes que nous avons obtenu concernent :

⇒ ***La surface occupée:***

- *Au fur et à mesure que le nombre de connexions et de transistors augmentent, le circuit devient moins dense. La densité des circuits ayant des portes de transmission est plus faible à cause des entrées non duales.*
- *L'augmentation de la densité des transistors en mm^2, par rapport à l'évolution des règles de dessin est proportionnelle au carré de la diminution de la longueur des transistors. Par exemple, entre la technologie ECPD15 et la technologie ECPD07 la densité moyenne est passée de 2130 T/mm^2 a 7982 T/mm^2.*
- *Le bénéfice de l'insertion de cassures dans les liaisons, lors du compactage des masques, a permis une amélioration moyenne de 7% dans la surface occupée.*
- *L'augmentation de surface est linéaire en fonction de la largeur des transistors.*
- *Le "rapport d'implantation" conseillé pour une macro-cellule doit être compris entre 0.5 et 2.*
- *Les cellules qui ont les meilleures performances électriques sont les multiplexeurs. Cependant, les cellules qui apportent le meilleur compromis temps-surface sont les portes complexes.*

⇒ ***Les performances électriques:***

- *Des simulations sur plusieurs circuits ont été effectuées, en fonction de la charge de sortie, les largeurs de transistors et la technologie. Ces simulations ont montré qu'à partir d'une certaine largeur de transistor, le retard du circuit devient presque constant par rapport à la charge de sortie. Dans la courbe du retard par rapport à la largeur des transistors, la région à partir de laquelle le retard est pratiquement constant est appelée " région stable".*

- *Le début de la région stable se situe à 10 fois le W minimal de la technologie (20 µm pour ECPD15, 15µm pour ECPD12, 12.5 µm pour ECPD10 et 10µm pour ECPD07)*

- *La comparaison entre la solution dimensionnée et la solution régulière a montré, pour le style linear-matrix, que la solution régulière est la plus performante en surface et temps. La solution dimensionnée fournit une solution initiale au compromis retard/puissance, permettant l'évaluation d'alternatives d'accélération à utiliser. La solution dimensionnée permet aussi de définir une largeur moyenne minimale pour l'implémentation régulière.*

- *L'utilisation de portes complexes permet de réduire le nombre de transistors, la surface occupée et le temps de propagation, par rapport à l'utilisation des portes simples.*

⇒ ***La comparaison d'approche de conception (standard-cells) :***

- *le retard du circuit peut être optimisé avec plus de souplesse, la migration technologique est simple, la surface occupée est moins importante et il n'y a pas de contraintes lors du "mapping".*

⇒ ***La comparaison de générateur (LAS) :***

- *le rapport d'occupation de surface entre TROPIC est LAS est en moyenne de 10%. Une des raisons à cette différence provient du routage vertical. Les deux générateurs permettent la migration technologique, du fait de l'utilisation d'une description symbolique. Cependant, TROPIC est plus simple à utiliser, il ne nécessite pas le coût d'un environnement aussi complexe que OPUS, et de plus il permet l'intégration avec des outils de dimensionnement des transistors.*

CHAPITRE 6

Conclusion générale

Dans ce mémoire nous avons présenté l'étude des méthodes de synthèse des masques des circuits intégrés. L'implémentation d'un générateur *linear-matrix* multi-bandes a permis l'analyse des performances temporelles, de la surface occupée et de la puissance dissipée des macro-cellules. Les principales caractéristiques du générateur sont la facilité de migration technologique, la possibilité de dimensionner les transistors et la non-utilisation de bibliothèques de cellules pré-caractérisées.

L'analyse de la surface occupée par les macro-cellules a été faite sur cinq jeux différents de règles de dessin. Les résultats montrent que la densité d'intégration est proportionnelle au carré de la diminution de la longueur des transistors. La surface occupée par un circuit est devenue ainsi un paramètre de deuxième ordre. Nous devons désormais faire attention principalement aux performances électriques (problèmes des résistances parasites en technologies submicroniques) et à la puissance dissipée.

L'utilisation des portes complexes a permis une réduction moyenne de 40% de la surface occupée et de 20% du temps critique des macro-cellules. Il reste maintenant a développer un module qui réalise de façon automatique l'insertion des portes complexes. Afin de contrôler les performances temporelles des circuits générés avec des portes complexes, le nombre maximal de transistors en série dans les plans 'N' et 'P' doit être un paramètre de ce module.

La figure 6.1 montre la synthèse des résultats des performances temporelles d'une macro-cellule, en fonction de la taille des transistors et de la charge de sortie. Pour des largeurs de transistors proches de la taille minimale de la technologie, le retard est très important et varie considérablement en fonction de la charge de sortie. Au fur et à mesure qui la taille des transistors augmente nous entrons dans une région stable, où le retard est presque constant en fonction de la charge de sortie. Le circuit dimensionné au début de cette région correspond au meilleur compromis entre la surface occupée, le retard et la puissance dissipée.

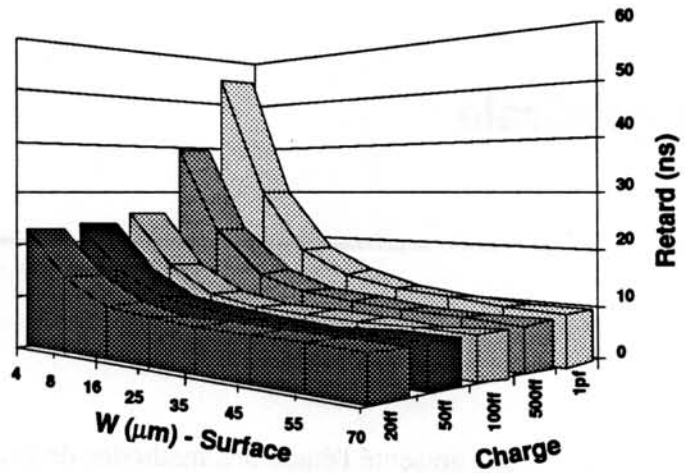


Figure 6.1 - Influence de la surface (W) et la charge de sortie sur le retard d'un circuit (technologie ECPD15)

Nous devons donc trouver la largeur des transistors correspondant au début de cette région stable. Une solution possible est le dimensionnement des transistors à dix fois la largeur minimale de la technologie. Nous avons montré que cette méthode est valable pour les cinq technologies étudiées. Nous avons remarqué par ailleurs que les bibliothèques de cellules pré-caractérisés industrielles utilisent cette méthode pour assurer un bon fonctionnement des circuits.

L'utilisation d'un outil de dimensionnement des transistors, comme PSIZE, assure un meilleur rapport entre les temps de montée et de descente. Dû à la dissymétrie de la taille des transistors, la surface occupée par la solution dimensionnée est plus importante que celle de la solution régulière. La solution proposée est l'utilisation d'une solution régulière, ayant la largeur moyenne des transistors de la solution dimensionnée. Si nécessaire, des étages tampons

sont ajoutés à la sortie des cellules fortement chargées.

Notre méthode permet non seulement de générer des macro-cellules, mais aussi de générer des circuits complets. Nous avons développé des procédures pour l'intégration des macro-cellules dans un environnement de CAO industriel, afin de réaliser les étapes du plan directeur et de routage global. Le résultat final est la génération de circuits complexes, ayant plus de 20.000 transistors, divisées en macro-cellules optimisées. La validation des performances temporelles de ces circuits étant réalisée par la simulation au niveau interrupteur.

L'intégration du générateur des macro-cellules avec les outils développés au laboratoire a abouti à un système performant de conception des circuits intégrés, utilisé par d'autres groupes de recherche ayant besoin de la synthèse automatique des masques, de la validation d'architectures ou de la caractérisation de blocs fonctionnels.

Suite à ce travail, les thèmes de recherche à développer sont principalement :

- Assignation technologique en utilisant de portes complexes (*AOI*). Cette procédure permettra la génération des circuits ayant de meilleures performances temporelles, avec une surface moins importante. Ce travail sera développé par M. André REIS, comme une partie de son doctorat.

- Partition automatique de circuits complexes, afin de déterminer les macro-cellules qui devront être générées.

- Intégration entre la synthèse de haut-niveau et la synthèse physique.

- Détermination des vecteurs de test qui exciteront le chemin critique d'un circuit donné, afin de déterminer les performances temporelles.

- Intégration efficace avec des outils de plan directeur, afin de déterminer la forme et la position des broches d'entrées/sorties des macro-cellules.

- Détermination des critères d'insertion d'étages tampons, afin d'éviter la perte en surface due aux différences des largeurs de transistors, et ainsi arriver à une solution régulière.

- Détermination de nouveaux styles de dessin des masques, en tenant compte des nouvelles technologies (3 niveaux de métallisation et contacts empilées).

- Génération automatique de circuits pour des hautes-performances, comme les logiques dynamiques.

REFERENCES BIBLIOGRAPHIQUES

[ABO92] **P.ABOUZEID, R.LEVEUGLE, G.SAUCIER, R.JAMIER**
"Logic Synthesis for Automatic Layout"
EUROASIC 92, pp. 146-151.

[AKE81] **S.B.AKERS**
"On the use of the Linear Assignment Algorithm in Module Placement"
18th ACM/IEEE Design Automation Conference, Nashville, 1981, pp. 137-143.

[AMA93] **S.AMAT**
"Contribution à l'assignement technologique contrôlé par les performances"
Thèse de doctorat, LIRMM, Montpellier, France, 1993.

[ANT92] **O.ANTON, K.DOERFFER, D.A.MLYNSKI**
"TRANSCAD: a CAD tool for transparent standard cells"
ISCAS 92, San Diego, 1992, pp. 1969-1972.

[APT87] **J.APTE, G.KEDEM**
"Strip layout: a new layout methodology for standard circuits modules"
24th ACM/IEEE Design Automation Conference, Miami Beach, 1987, pp. 363-369.

[AUV90] **D.AUVERGNE, D.DESCHACHT, M.ROBERT**
"Input wave form slope effects in CMOS delays"
IEEE Journal of Solid State Circuits, Vol. SC-25, No. 6, December. 1990.

[AUV91] **D.AUVERGNE, N.AZEMARD, V.BONZOM, D.DESCHACHT, M.ROBERT**
"Formal sizing rules of CMOS circuits"
EDAC 91, Amsterdam, 1991, pp. 96-100.

[AZE92] **N.AZEMARD, V.BONZOM, D.AUVERGNE**
"P.SIZE : a sizing aid for optimized designs"
EURO-DAC 92 , Hamburg, 1992, pp. 160-165.

[BAL88] **D.G.BALTUS, J.ALLEN**
"SOLO: a generator of efficient layouts from optimized MOS circuit schematics"
25th ACM/IEEE Design Automation Conference, Anaheim, 1988, pp. 445-452.

[BHI93] **S.BHINDARDE, A.PANYAM, N.A.SHERWANI**
"On optimum cell models for over-the-cell routing"
6th International Conference on VLSI Design, Bombay, 1993, pp. 94-99.

[BON91] **P.BONZOM**
"Dimensionnement automatique de structures MOS pour générateur contrôle par les performances"
Thèse de doctorat, LIRMM, Montpellier, France, 1991.

[BRE77] **M.A.BREUER**
"A class of Min-Cut Placement Algorithms"
14th ACM/IEEE Design Automation Conference, New Orleans, 1977, pp. 284-290.

[BRG89] **F.BRGLEZ, D.BRYAN, K.KOZMINSKI**
"Combinational profiles of sequential benchmark circuits"
ISCAS 89, 1992, pp. 1929-1934.

[CAD89] **CADENCE™**
"EDGE Integrated IC Design System - Reference Manuals", 1989.

[CAD91] **CADENCE™**
"HDL Synthesizer and Optimizer - USER GUIDE"
Version 4.2, October 1991.

[CAD93] **CADENCE™**
"Switch-RC Technology Characterization",
June 1993, pp. C1-C48.

[CAR92] **B.S.CARLSON, C.Y.R.CHEN, D.S.MELIKSETIAN**
"An efficient algorithm for the identification of dual eulerian graphs and its application to the cell layout"
ISCAS 92, San Diego, 1992, pp. 2248-2251.

[CAT90] **G.CATHEBRAS**
"Contribution à la compilation structurelle des circuits intégrés CMOS"
Thèse de doctorat, LIRMM, Montpellier, France, 1990.

[CC88] **C.Y.R.CHEN, C.Y.HOU**
"A new layout optimization methodology for CMOS complex gates"
ICCAD 88, Santa Clara, 1988, pp. 368-371.

[CC89] **C.C.CHEN, S.L.CHOW**
"The layout synthetizer: an automatic netlist-to-layout system"
26th ACM/IEEE Design Automation Conference, Las Vegas, 1989, pp.232-238.

[CHA87] **Y.CHANG, S.C.CHANG, L.H.HSU**
"Automated layout generation using gate matrix approach"
24th ACM/IEEE Design Automation Conference, Miami Beach, 1987, pp.552-558.

[CHA90] **E. CHAURAND**
"Placement des blocs de cellules standard par bissections parallélisées avec routage global simultané"
Thèse de doctorat, Université de Paris Sud (Orsay), Janvier 1990.

[CH88] **H.Y.CHEN, S.M.KANG**
"iCOACH: a circuit optimization aid for CMOS high-performance circuits"
ICCAD 88, Santa Clara, 1988, pp. 372-375.

[CHE91] **C.F.CHEN, C. LO, H.N.NHAM, P.SUBRAMANIAM**
"The second generation MOTIS mixed mode simulation"
21th ACM/IEEE Design Automation Conference, Albuquerque, 1984.

[CON90] **J.CONG, B.T.PREAS, C.L.LIU**
"General Models and algorithms for the over-the-cell routing in standard-cell design"
27th ACM/IEEE Design Automation Conference, Orlando, 1990, pp. 709-715.

[COU92] **C.COUNIL, M.RENOVELL, G. GAMBON**
"Self-Test for F.I.R. Filters"
Workshop on VLSI Signal Processing, Napa, 1992, pp. 51-60.

[DES88] **D.DESCHACHT, M.ROBERT, D.AUVERGNE**
"Explicit formulation of delays in CMOS data path"
IEEE Journal of Solid State Circuits, vol. 23, no 5, October. 1988, pp. 1257-1264.

[DES90] **D.DESCHACHT, P.PINEDE, M.ROBERT, D.AUVERGNE**
"PATH RUNNER: an accurate and fast timing analyser"
EDAC 90, Glasgow, 1990, pp. 529-533.

[DET87] **E.DETJENS, G.GANNOT, R.RUDELL, S.VINCENTELLI, A.WANG**
"Technology mapping in MIS"
ICCAD, Santa Clara, 1987, pp. 116-119.

[DEU85] **D.N.DEUTSCH**
"Compacted channel routing"
ICCAD 85, pp. 223-225.

[DUF90] **J.C.DUFOURD, J.F.NAVINER, F.JUTAND**
"Preform : A Process Independent Symbolic Layout System"
ICCAD 90, pp. 248-251.

[DUN85] **A.E.DUNLOP, B.W.KERNIGHAN**
"A procedure for placement of standard-cell VLSI circuits"
IEEE Transactions on Computer Aided Design, Vol. CAD-4, No.1, January 1985, pp. 92-98.

[ESB92] **H.ESBENSEN**
"A genetic algorithm for macro cell placement"
EURO-DAC 92 , Hamburg, 1992, pp. 52-57.

[FID82] **C.M.FIDUCCIA, R.M.MATTHEYSES**
"A linear time heuristic for improving network partitions"
19th ACM/IEEE Design Automation Conference, Las Vegas, 1982, pp. 175-181.

[FIS85] **P.FISHBURN, A.E.DUNLOP**
"TILOS: A Posynomial Programming Approach to Transistor Sizing"
ICCAD, Santa Clara, 1985, pp. 326-328.

[GRE92] **A.GREINER, F.PECHEUX**
"ALLIANCE : A Complete Set of CAD Tools for Teaching VLSI Design"
3th EUROCHIP Workshop on VLSI Design Training, 1992, pp. 230-237.

[H90] **META-SOFTWARE**
"HSPICE User's Manual H9001", 1990.

[HAS71] **A.HASHIMOTO, J.STEVENS**
"Wire routing by optimizing channel assignment within large apertures"
8th Design Automation Workshop, 1971, pp. 155-163.

[HEE92] **H.HEEB, W.FICHTNER**
"A module generator based on the PQ-tree algorithm"
IEEE Transactions on Computer-Aided Design, Vol. 11, No.7, July 1992, pp. 876-884

[HEN85] **B.HENNION, P.SENN**
"ELDO : a new third generation circuit simulator using the one step relaxation method"
ISCAS 85, Kyoto, 1985.

[HOF87] **M. HOFMANN, J.K. KIM**
"Delay optimization of combinational static CMOS logic"
24th ACM/IEEE Design Automation Conference, Miami Beach, 1987, pp. 125-132.

[HSI91] **Y.HSIEH, C.HWANG, Y.LIN, Y.HSU**
"LiB: a CMOS Cell Compiler"
IEEE Transactions on Computer-Aided Design, Vol. 10, No. 8, August 1991, pp. 994-1005.

[HWA89] **C.HWANG, Y.HSIEH, Y.LIN, Y.HSU**
"An optimal transistor-chaining algorithm for CMOS cell layout"
ICCAD, Santa Clara, 1989, pp. 344-347.

[HWA90] **C.HWANG, Y.HSIEH, Y.LIN, Y.HSU**
"A fast transistor-chaining algorithm for CMOS cell layout"
IEEE Transactions on Computer-Aided Design, Vol. 9, No. 7, July 1990, pp. 781-786.

[HWA93] **C.HWANG, Y.HSIEH, Y.LIN, Y.HSU**
"An efficient layout style for two-metal CMOS leaf cells and its automatic synthesis"
IEEE Transactions on Computer-Aided Design of ICs and Systems, Vol. 12, No.3, March 1993, pp. 410-423.

[KAM82] **T.KAMBE, T.CHIBA, S.KIMURA, T.INUFUSHI**
"A placement algorithm for polycell LSI and its Evaluation"
19th ACM/IEEE Design Automation Conference, Las Vegas, 1982, pp. 655-662.

[KAN83] **S.KANG**
"Linear Ordering and Application to Placement"
20th ACM/IEEE Design Automation Conference, Miami Beach, 1983, pp. 457-464.

[KER70] **B.W.KERNIGHAN, S.LIN**
"An efficient heuristic procedure for partitioning graphs"
Bell System Technical Journal, No. 49, February 1970, pp. 291-308.

[KIM92a] **S.KIM, R.M.OWENS, M.J.IRWIN**
"Experiments with a performance driven module generator"
29th ACM/IEEE Design Automation Conference, 1992, pp. 687-690.

[KIM92b] **S.KIM, R.M.OWENS, M.J.IRWIN**
"PERFLEX : a performance driven module generator"
EURO-DAC 92 , Hamburg, 1992, pp. 154-159.

[KIR83] **S.KIRKPATRICK, C.D.GELLAT, M.P.VECCHI**
"Optimization by simulated annealing"
Science, 220, May 1983, pp. 671-680.

[KOL85] **P.W.KOLLARITSCH, N.WESTE**
"TOPOLOGIZER: an expert system translator of transistor connectivity to symbolic cell layout"
IEEE Journal of Solid State Circuits, Vol. SC-20, No.3, June 1985, pp. 779-804.

[KOZ83] **T.KOZAWA** et al.
"Automatic Placement Algorithms for High Packing Density VLSI"
20th ACM/IEEE Design Automation Conference, Miami Beach, 1983, pp. 175-181.

[KWO88] **Y.KWON, C.KYUNG**
"A fast heuristics for optimal CMOS functional cell layout generation"
ISCAS 88, Helsinki, 1988, pp. 2423-2426.

[LAK90] **G.LAKHANI, S.RAO**
"A multiple row-based layout generator for CMOS cell"
ISCAS 90, New Orleans, 1990, pp. 1697-1700.

[LEF89] **M.LEFEBVRE, C.CHAN**
"Optimal ordering of gate signals in CMOS complex gates"
CICC 89, pp. 17.5.1-17.5.4.

[LIN87] **S.L.LIN, J.ALLEN**
"LES : a layout expert system"
24th ACM/IEEE Design Automation Conference, Miami Beach, 1987, pp. 672-678.

[LIN91] **M.LIN, H.PERNG, C.HWANG, Y.LIN**
"Channel density reduction by routing over the cells"
28th ACM/IEEE Design Automation Conference, San Francisco, 1991, pp. 120-125.

[LIN92] **H.LIN, H.PERNG, Y.HSU**
"Cell height reduction by routing over the cells"
ISCAS 92, San Diego, 1992, pp. 2244-2247.

[LOP80] **A.D.LOPEZ, H.S.LAW**
"A dense gate matrix layout for MOS VLSI"
IEEE Transactions on Electron Device, Vol. ED-27, No. 8, August 1980, pp. 1671-1675.

[LUB90] **M.LUBASZEWSKI**
"Geração automática de lógica aleatória utilizando a metodologia TRANCA"
"Master Thesis", UFRGS-CPGCC, Porto Alegre, Brésil, 1990.

[MAD89] **J.MADSEN**
"A new approach to optimal cell synthesis"
ICCAD, Santa Clara, 1989, pp. 336-339.

[MAR89] **D. MARPLE**
"Transistor size optimization in the Tailor layout system"
26th ACM/IEEE Design Automation Conference, Las Vegas, 1989, pp. 43-48.

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

[MAS90] **C.MASSON, D.BARBIER, R.ESCASSUT, D.WIMER, G.CHEVALLIER, P.F.ZEEGERS**
"CHEOPS : an integrated VLSI floor planning and chip assembly system implemented in object oriented LISP"
EDAC 90, Glasgow, 1990, pp.250-261.

[MAZ87] **R.L.MAZIASZ, J.P.HAYES**
"Layout optimization of CMOS functional cells"
24th ACM/IEEE Design Automation Conference, Miami Beach, 1987, pp. 554-551.

[MAZ91] **R.L.MAZIASZ, J.P.HAYES**
"Exact width and height minimization of CMOS cells"
28th ACM/IEEE Design Automation Conference, San Francisco, 1991, pp. 487-493

[MAZ92] **R.L.MAZIASZ, J.P.HAYES**
"Layout minimization of CMOS Cells"
Kluwer Academic Publishers, 1992.

[MOR90] **F.MORAES, R.REIS**
"EXTRALO - Extrator Lógico"
5th Simpósio Brasileiro de Concepção de Circuitos Integrados, Ouro Preto (Brésil), 1990, pp. 167-176

[MOR90b] **F.MORAES**
"TRAGO - Síntese Automática de Leiaute para Circuitos em Lógica Aleatória"
"Master Thesis", UFRGS-CPGCC, Porto Alegre, Brésil, 1990.

[MOR93] **F.MORAES, N.AZEMARD, M.ROBERT, D.AUVERGNE**
"Flexible Macrocell Layout Generator"
4th ACM/SIGDA Physical Design Workshop, Los Angeles, 1993, pp. 105-116.

[NAK92] **T.NAKAGAKI, S.YAMADA, K.FUKUNAGA**
"Fast optimal algorithm for the CMOS functional cell layout based on transistor reordering"
ISCAS 92, San Diego, 1992, pp. 2116-2119.

[NID93] **F.NIDEGGER**
"Etude de la conception des multiplieurs rapides"
Rapport de DEA, LIRMM, Montpellier, France, 1993.

[OBE88] **F.W.OBERMEIER, R.KATZ**
"An electrical optimizer that consider physical layout"
25th ACM/IEEE Design Automation Conference, Anaheim, 1988, pp. 453-459.

[OCT91] **OCTTOOLS 5.1**
"User guide"
University of California, Berkley, 1991.

[ONG89] **C.ONG, J.LI, C.LO**
"GENAC: an automatic cell synthesis tool"
26th ACM/IEEE Design Automation Conference, Las Vegas, 1989, pp. 239-244

[ORE93] **Y. ORER**
"Comparaison d'opérateurs arithmétiques - structures d'additionneurs"
Rapport de DEA, LIRMM, Montpellier, France, 1993

[PAT81] **A.M.PATEL**
"Partitioning for VLSI placement problems"
18th ACM/IEEE Design Automation Conference, Nashville, 1981, pp. 411-418

[PED91] **M.PEDRAM, N.BHAT**
"Layout driven technology mapping"
28h ACM/IEEE Design Automation Conference, San Francisco, 1991, pp. 99-105

[PIG88] **C.PIGUET** et al.
"Alladin: a CMOS gate matrix layout system"
ISCAS 88, Helsinki, pp. 2427-2430.

[PIL88] **L.T.PILLAGE, R.A.ROHRER**
"A quadratic metric with a simple solution scheme for initial placement"
25th ACM/IEEE Design Automation Conference, Anaheim, 1988, pp. 324-329

[PRE88] **B.T.PREAS, M.J.LORENZETTI**
"Physical Design Automation of VLSI Systems"
Benjamin/Cummings Publishing, 1988.

[REI83] **R.REIS**
"TESS: evaluateur topologique predictif pour la génération automatique des plans de masse de circuits VLSI"
Thèse de doctorat, Institut Polytechnique de Grenoble, Grenoble, France, 1983.

[REI87] **R.REIS**
"A new standard cell methodology"
CICC, Portland, 1987, pp. 4-7.

[REI88] **R.REIS, R.GOMES, M.LUBASZEWSKI**
"An efficient design methodology for standard cell circuits"
ISCAS 88, Helsinki, pp. 1213-1216.

[RIV82] **R.L.RIVEST, C.M.FIDUCCIA**
"A 'greedy' channel router"
19th ACM/IEEE Design Automation Conference, Las Vegas, 1982, pp. 208-219.

[ROB88] **M.ROBERT, D.DESCHACHT, G.CATHEBRAS, D.AUVERGNE, S.PRAVOSSOUDOVITCH**
"PRINT Methodology: a compilation approach for cell library generation"
ISCAS 88, Helsinki, pp. 965-968.

[ROB91] **M.ROBERT, J.TRAUCHESSEC, G.CATHEBRAS, V.BONZOM, N.AZEMARD, D.DESCHACHT, D.AUVERGNE**
"Evaluation of VLSI layout style implementations for efficiency "
EUROASIC 91, pp. 362-365

[ROB94] **M.ROBERT, L.TORRES, F.MORAES, D.AUVERGNE**
"Influence of Logic Block Layout Architecture on FPGA Performance"
4th International Workshop on FPGA (FPL '94), Prague, 1994.

[SCH72] **D.M.SCHULER, E.G.ULRICH**
"Clustering and linear placement"
9th Design Automation Workshop, 1972, pp. 50-56.

[SCH76] **D.G.SCHWEIKERT**
"A 2-dimensional placement algorithm for the layout of electrical circuits"
13th ACM/IEEE Design Automation Conference, 1976, pp. 408-416.

[SHY88] **J.M.SHYU, A.SANGIOVANNI-VINCENTELLI, J.P.FISHBURN, A.E. DUNLOP**
"Optimization based transistor sizing"
IEEE Journal of Solid State Circuits, Vol. SC-23, No.2, April 1988, pp. 400-409.

[SZY85] **T.G.SZYMANSKI**
"Dogleg channel routing is NP-complete"
IEEE Transactions on Computer Aided Design, Vol. CAD-4, No. 1, January 1985, pp. 31-41.

[THO91] **D.E.THOMAS, P.R.MOORDY**
"The Verilog Hardaware Description Language"
Kluwer Academic Publishers, 1991.

[TRA91] **J.TRAUCHESSEC**
"Contribution à la synthèse topologique de circuits integrés CMOS"
Thèse de doctorat, LIRMM, Montpellier, France, 1991.

[UEH81] **T.UEHARA, W.VANCLEEMPUT**
"Optimal layout of CMOS functional arrays"
IEEE Transactions on Computers, Vol. C-30, No. 5, May 1981, pp. 305-312.

[VIR91] **VIRTUOSO LAYOUT SYNTHESIZER User Guide - CADENCE™**
Version 4.2 October 1991.

[VLA80] **VLADIMIRESCU, S.LIU**
"The simulation of MOS integrated circuits using SPICE2"
University of California, Berkeley, February 1980.

[WEI67] **A.WEINBERGER**
"Large scale integration of MOS complex logic: a layout method"
IEEE Journal of Solid State Circuits, Vol. SC-2, No. 4, December 1967, pp.182-190.

[WIN82] **O.WING**
"Automated gate matrix layout"
ISCAS 82, Rome, pp. 681-685.

[WIN87] **S.WIMMER, R.Y.PINTER, J.A.FELDMAN**
"Optimal chaining of CMOS transistors in a functional cell"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No.5, September 1987, pp. 795-801.

TABLE DES FIGURES

TABLE DES FIGURES

FIGURE CHAPITRE 1

FIGURES CHAPITRE 2

FIGURES CHAPITRE 3

FIGURES CHAPITRE 4

FIGURES CHAPITRE 5

FIGURES CHAPITRE 6

ANNEXE 1

Guide d'utilisation du système TROPIC

Nous présentons le fichier "README", qui explique à l'utilisateur les principales procédures pour générer une macro-cellule.

Notice d'utilisation de TROPIC-PRINT

FICHIER : ~moraes/tropic/README-tropic

Mise à jour précédente : 03/11/93
Date du mise a jour : 11/04/94

Version du logiciel TROPIC : version 4.0 (exécutable : tr9)

Version précédente : version 3.5 (exécutable : tr8)

POUR INSTALLER LE SYSTÈME TROPIC

-> DANS LE RÉPERTOIRE RACINE (de login):
taper : *bar -Zxvf ~moraes/tropicFILES*
en utilisant 'vi', inclure dans le fichier '.cshrc' la ligne: *set path = (~moraes/bin $path)*
taper : *source ~/.cshrc*

-> ALLER DANS LE RÉPERTOIRE DE TRAVAIL
taper: *cd tropic*
choisir la technologie, taper : *settec tecXX (XX étant 20, 15, 12, 10, 07)*

RESTRICTIONS DU SYSTÈME

* UNIQUEMENT LOGIQUE COMPLÉMENTAIRE ET PORTES DE TRANSMISSION.
* NE SONT PAS PERMIS: logique dynamique, logique DOMINO, logique NORA, ...

LIAISON AVEC LES AUTRES OUTILS

Le module de génération de masques est associe avec un outil d'extraction électrique et un outil de dimensionnement automatique de transistors :

tr9 - générateur de masques a partir d'une description SPICE (TROPIC)
net - extracteur électrique avec parasites (EDGE)
sizing - dimensionnement de transistors (PSIZE : N.AZEMARD)

Par défaut, le compacteur utilisé dans tr9 est le compacteur PRINT, développe par G.CATHEBRAS.

TECHNOLOGIES DISPONIBLES

- utilisation du compacteur PRINT (défaut) : technologies ES2 2µm et 1.5µm.

- utilisation du compacteur EDGE: technologies ES2 2.0, 1.5, 1.2, 1.0 et 0.7 µm.

EXEMPLE D'UTILISATION

1) Aller dans le répertoire où se trouvent les fichiers TROPIC => ***cd tropic***

2) Générer les masques du circuit à partir de la netlist SPICE:

Compactage PRINT : ***tr9 adder.sim***

Compactage EDGE :
taper: ***tr9 adder.sim*** /* génère seulement la description symbolique */
syl2cad adder /* conv. symbolique vers la base de donnes EDGE */
ensuite appeler EDGE et faire le compactage

3) Pour voir les masques, on appelle EDGE => taper ***opencds***

- dans EDGE, sélectionner 'view' et le circuit 'adder' 'layout'
- la surface de la cellule est le rectangle qui enveloppe la cellule, sans les deux lignes du peigne d'alimentation

4) Quitter EDGE

5) Pour faire l'extraction électrique avec parasites taper ***net -t 15 adder.sim*** => résultat : fichier adder.net

TRÈS IMPORTANT :
- le résultat de l'extraction électrique est le fichier *<circuit>.net*
- ce fichier contient les capacités parasites, la carte model avec les paramètres de la technologie, les transistors et les interfaces.
- la carte model par défaut est le niveau 2 typique, fournit par ES2. Pour d'autres niveaux de modélisation (level 6, par exemple) il faut changer la carte model dans le fichier *<circuit>.net*
- le paramètre 't' spécifie la technologie (2, 15, 12, 10, 07)

6) Pour faire le dimensionnement des transistors taper: ***sizing -s adder.cir new.sim***

- cette procédure, sizing, va filtrer les labels des équipotentielles en les remplaçant par de numéros (compatibilité avec le format standard SPICE 2G.6). Pour savoir la correspondance entre les labels et les numéros voir le fichier TABLE.
- la netlist 'new.sim' contiendra les transistors dimensionnes
- on pourra refaire la génération des masques a partir de cette netlist, en tapant '***tr9 new.sim***'
- pour plus d'information voir la notice d'utilisation de P.SIZE.

7) La largeur moyenne du circuit dimensionné est donnée par la commande : wm <nom du fichier>
exemple : ***wm new.sim***

8) Pour faire la simulation électrique taper : ***hspice adder.cir***

- le résultat de la simulation est le fichier adder.tr0, avec toutes les courbes de la simulation. Pour regarder les courbes, on utilise le logiciel GSI (graphic standard interface) ou HSPLOT. Pour cela regarder la documentation spécifique de HSPICE.

9) Optionnel:

- pour réaliser l'extraction logique, c'est a dire, trouver toutes les cellules qui sont dans une netlist , on utilise la commande 'extralo'. Exemple: ***extralo adder.net f_out***

- adder.net est le fichier d'entrée, à plat (sans sous-circuits), et "f_out" est le fichier de sortie, contenant les portes de base de adder.net.

RÈGLES POUR UNE NETLIST CORRECTE

- L'utilisateur doit avoir pour chaque circuit deux fichiers:
  <circuit>.sim : netlist SPICE pour la génération de masques
  <circuit>.cir : netlist SPICE pour la simulation électrique

  Voir les exemples: *adder.sim* et *adder.cir*

- Pour utiliser les portes existantes dans la bibliothèque il suffit d'inclure au début du fichier *<circuit>.sim* la commande '.include librairie'.

- Portes disponibles dans la bibliothèque structurelle
  * inverseur - inv
  * distribution d'horloge - arbre
  * mux 2-1 - mux2to1
  * nxor 2 entrées - nxor
  * and2, and3, and4, and5
  * or2, or3, or4
  * bascule d avec clear - dffclear
  * demi-additionneur - demi-add
  * adder 4 bits carry look-ahead - adder4
  * buffer - buf
  * porte de transmission - trg
  * xor 2 entrées - xor
  * nand2, nand3, nand4, nand5
  * nor2, nor3, nor4
  * bascule d - dff
  * demi-additionneur plus 1 - add1
  * full-adder 2 bits - adder

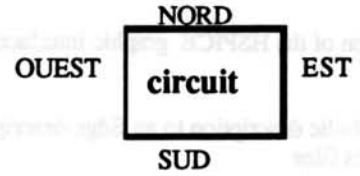
- Exemple de description en utilisant la librairie: adder.sim

```
.include librairie

X1  a b c carry sum vcc adder

*interface: a          orient o
*interface: b          orient o
*interface: c          orient o
*interface: carry      orient s
*interface: sum        orient s
```

- Il faut toujours déclarer l'orientation des broches d'entrées/sorties:



Syntaxe: *interface <nom du pin> orient <'O' ou 'E' ou 'N' ou 'S'>
Exemple: **interface clock orient N*

- Déclarer les sous-circuits avant de les utiliser. Exemple:

```
                .subckt inv                    X1 a b vcc inv
                ........
CORRECT         .endsinv          INCORRECT    .subcktinv
                                               ........
                X1 a b vcc inv                 .endsinv
```

- Le système ne fait pas la différence entre lettres minuscules et majuscules, mais il est fortement recommandé d'utiliser les minuscules, pour éviter des problèmes avec EDGE et HSPICE.

- Le signal de masse doit être déclare comme "gnd" et le signal d'alimentation "vcc".

- N'utiliser pas les grilles des transistors connectées à "gnd" ou "vcc". Si une entrée d'une porte doit être connectée à "gnd" ou "vcc" le plus raisonnable est d'utiliser d'autres porte, avec un nombre plus petit d'entrées.

PARAMÈTRES OPTIONNELS DE TROPIC (TR9)

tr9 -c -S -t <technologie> -b <n> -w <n> -l <n> -r <rules file> input file

-c :	génère seulement la description symbolique, sans compactage.
-S :	mettre la même largeur pour tous les transistors de la même cellule.
-t n :	spécification de la technologie (2, 15, 12, 10, 07).
-b n :	spécification du nombre 'n' de lignes de cellules (bandes).
-w n :	spécification de la largeur (W) 'n' pour tous les transistors du circuit, en microns.
-l n :	spécification de la longueur (L) 'n' pour tous les transistors du circuit, en microns.
-r <rules file> :	spécifie le nom du fichier de paramètres, par défaut, le fichier est le 'rul.cfg', qui est dans racine du utilisateur.

- Exemple: si on veut générer le circuit 'test.sim' avec 3 bandes, tous les transistors avec w=10µm, l=1µm, dans la technologie ECPD10 (1µm) :

tr9 -b 3 -w 10 -l 1 -t 10 test.sim

- pour voir les paramètres de chaque module il suffit de taper son nom, par exemple: *'tr9'* ou *'net'* ou *'sizing'* ou *'extralo'*.

BUGS DÉTECTÉS

- quand le circuit est constitue avec beaucoup de portes de transmission les "body-ties" risquent de ne pas être placés. La conséquence sera, au moment de l'extraction électrique, l'apparition de capacités fantômes. Pour la simulation, enlever ces capacités fantômes du fichier extract.
- si les transistors ont des largeurs 'W' différentes dans la même cellule, il est très probable que le compactage PRINT échoue. Pour résoudre il suffit de générer la cellule avec l'option "-s"

FICHIERS NÉCESSAIRES AU SYSTÈME

Configuration :	~/.bindkey	- EDGE bindkeys
	~/.sdarc	- EDGE configuration
	~/rul.cfg	- TROPIC configuration
	~/meta.cfg	- HSPICE configaration
	~/gsi.Xdefaults	- Xwindows configuration of the HSPICE graphic interface
	.sdalocal	- EDGE configuration
	layers	- EDGE layers
Outils :	syl2cad	- conversion from a symbolic description to an Edge description
	settec	- batch to install the rules files
Exemples:	librarie	- subcircuits library
	*.sim	- spice exemples
	adder.cir	- spice file for simulation
	adder	- layout exemple

UTILISATION DU COMPACTEUR DE CADENCE (EDGE)

(extrait de la notice d'utilisation)

- To setup the directory in order to run cadence compactor, you must compile the technology file. This is done by the cshell script "*settec*" of the current directory. This script expect the name of the technology (tec20, tec15, tec12, tec10 or tec07).
- Exemple : to set the current technology to ecpd15 : *settec tec15*
- To convert a symbolic description to edge data base : *syl2cad <circuit name>*
- After this you can open the cadence design framework (in this directory to get the configuration).
- The first step is to compact the symbolic description to a "compacted" one. This is done by the "compact" menu.
- The conversion of the "compacted description" into a layout is done using the "1" bindkey (this is set in the .sdalocal file). This run an IL script which asks you the name of the "compacted" rep, the name of the "layout" rep and the name of the IL conversion file. Defaults are often the best choice ! As for the compaction, you are asked for pointing a window to place the layout rep.

ANNEXE 2

Bibliothèque structurelle

Nous montrons ci-dessous le fichier contenant la liste de portes utilisées pour la définition d'une description SPICE hiérarchisée.

```
* liste de sous-circuits, pour l'utiliser inserer dans le
* fichier  topologie, au début, la commande
* .include librarie


.subckt inv in out vcc
MP1  out  in vcc vcc      PMOS l=1.6U w=12.0U
MN2  out  in gnd gnd      NMOS l=1.6U w=12.0U
.ends inv

.subckt buf in out vcc
MP1  out  in vcc vcc      PMOS l=1.6U w=12.0U
MP2  out  in vcc vcc      PMOS l=1.6U w=12.0U
MN3  out  in gnd gnd      NMOS l=1.6U w=12.0U
MN4  out  in gnd gnd      NMOS l=1.6U w=12.0U
.ends buf

.subckt arbre in o1 o2  vcc
X1       in _2     vcc      inv
X2       _2 o1     vcc      buf
X3       in o2     vcc      buf
.ends arbre

.subckt trg in out h nh vcc
M1 in   h  out gnd        NMOS l=1.6U w=12.0U
M2 in   nh  out vcc       PMOS l=1.6U w=12.0U
 ends trg

.subckt latch in q q_ h nh vcc
M1 a   q_ vcc vcc         PMOS l=1.6U w=12.0U
M2 a   q_ gnd gnd         NMOS l=1.6U w=12.0U
M3 q_   q  vcc vcc        PMOS l=1.6U w=12.0U
M4 q_   q  gnd gnd        NMOS l=1.6U w=12.0U
M5 a   nh q   gnd         NMOS l=1.6U w=12.0U
M6 a   h  q   vcc         PMOS l=1.6U w=12.0U
M7 in  h  a   gnd         NMOS l=1.6U w=12.0U
M8 in  nh a   vcc         PMOS l=1.6U w=12.0U
.ends latch

.subckt mux2to1 i1 i2 C out vcc
X1 C nc vcc               inv
X2 i1 out C  nc vcc       trg
X3 i2 out nC C  vcc       trg
.ends mux2to1

.subckt xor i1 i2 out vcc
MP1     _3 i1 vcc vcc     PMOS l=1.6U w=12.0U
MP2     _4  i2 _3  vcc    PMOS l=1.6U w=12.0U
MP3     vcc  i1 _5  vcc   PMOS l=1.6U w=12.0U
MP4     vcc  i2 _5  vcc   PMOS l=1.6U w=12.0U
MP5     _5  _4 out vcc    PMOS l=1.6U w=12.0U
MN6     _4  i1 gnd gnd    NMOS l=1.6U w=12.0U
MN7     _4  i2 gnd gnd    NMOS l=1.6U w=12.0U
MN8     out  _4 gnd gnd   NMOS l=1.6U w=12.0U
MN9     _6  i1 gnd gnd    NMOS l=1.6U w=12.0U
MN10 out  i2 _6  gnd      NMOS l=1.6U w=12.0U
.ends xor

.subckt nxor a b out vcc
MP1  _3  a  vcc vcc       PMOS l=1.6U w=12.0U
MP2 out  b  _3  vcc       PMOS l=1.6U w=12.0U
MP3 out  y  vcc vcc       PMOS l=1.6U w=12.0U
MN4  _4  y  out gnd       NMOS l=1.6U w=12.0U
MN5  _4  a  gnd gnd       NMOS l=1.6U w=12.0U
MN6  _4  b  gnd gnd       NMOS l=1.6U w=12.0U
MP7     y  a  vcc vcc     PMOS l=1.6U w=12.0U
MP8     y  b  vcc vcc     PMOS l=1.6U w=12.0U
MN9     y  a  _5  gnd     NMOS l=1.6U w=12.0U
MN10 _5  b  gnd gnd       NMOS l=1.6U w=12.0U
.ends nxor

.subckt nand2 i1 i2 out vcc
MP1  out  i1 vcc vcc      PMOS l=1.6U w=12.0U
MP2  out  i2 vcc vcc      PMOS l=1.6U w=12.0U
MN3  out  i1 _2  gnd      NMOS l=1.6U w=12.0U
MN4  _2   i2 gnd gnd      NMOS l=1.6U w=12.0U
.ends nand2
```

```
.subckt nand3 i1 i2 i3 out vcc
MP1  out  i1 vcc vcc      PMOS l=1.6U w=12.0U
MP2  out  i2 vcc vcc      PMOS l=1.6U w=12.0U
MP3  out  i3 vcc vcc      PMOS l=1.6U w=12.0U
MN4  out  i1 _4  gnd      NMOS l=1.6U w=12.0U
MN5 _4    i2 _2  gnd      NMOS l=1.6U w=12.0U
MN6 _2    i3 gnd gnd      NMOS l=1.6U w=12.0U
.ends nand3

.subckt nand4 i1 i2 i3 i4 out vcc
MP1    out  i1 vcc vcc    PMOS l=1.6U w=12.0U
MP2    out  i2 vcc vcc    PMOS l=1.6U w=12.0U
MP3    out  i3 vcc vcc    PMOS l=1.6U w=12.0U
MP4    out  i4 vcc vcc    PMOS l=1.6U w=12.0U
MN5    out  i1 _4  gnd    NMOS l=1.6U w=12.0U
MN6    _4   i2 _3  gnd    NMOS l=1.6U w=12.0U
MN7    _3   i3 _2  gnd    NMOS l=1.6U w=12.0U
MN8    _2   i4 gnd gnd    NMOS l=1.6U w=12.0U
.ends nand4

.subckt nand5 i1 i2 i3 i4 i5 out vcc
MP1  out  i1 vcc vcc      PMOS l=1.6U w=12.0U
MP2  out  i2 vcc vcc      PMOS l=1.6U w=12.0U
MP3  out  i3 vcc vcc      PMOS l=1.6U w=12.0U
MP4  out  i4 vcc vcc      PMOS l=1.6U w=12.0U
MP5  out  i5 vcc vcc      PMOS l=1.6U w=12.0U
MN6  out  i1 _4  gnd      NMOS l=1.6U w=12.0U
MN7  _4   i2 _3  gnd      NMOS l=1.6U w=12.0U
MN8  _3   i3 _2  gnd      NMOS l=1.6U w=12.0U
MN9  _2   i4 _1  gnd      NMOS l=1.6U w=12.0U
MN10 _1   i5 gnd gnd      NMOS l=1.6U w=12.0U
.ends nand5

.subckt and2 i1 i2 out vcc
X1 i1 i2 _1               vcc    nand2
X2 _1 out                 vcc    inv
.ends and2

.subckt and3 i1 i2 i3 out vcc
X1 i1 i2 i3 _1            vcc    nand3
X2 _1 out                 vcc    inv
.ends and3

.subckt and4 i1 i2 i3 i4 out vcc
X1 i1 i2 i3 i4 _1         vcc    nand4
X2 _1 out                 vcc    inv
.ends and4

.subckt and5 i1 i2 i3 i4 i5 out vcc
X1 i1 i2 i3 i4 i5 _1      vcc    nand5
X2 _1 out                 vcc    inv
.ends and5

.subckt nor2 i1 i2 out vcc
MP1  _2   i1 vcc vcc      PMOS l=1.6U w=12.0U
MP2  out  i2 _2  vcc      PMOS l=1.6U w=12.0U
MN3  out  i1 gnd gnd      NMOS l=1.6U w=12.0U
MN4  out  i2 gnd gnd      NMOS l=1.6U w=12.0U
.ends nor2

.subckt nor3 i1 i2 i3 out vcc
MP1  _2   i1 vcc vcc      PMOS l=1.6U w=12.0U
MP2  _3   i2 _2  vcc      PMOS l=1.6U w=12.0U
MP3  out  i3 _3  vcc      PMOS l=1.6U w=12.0U
MN4  out  i1 gnd gnd      NMOS l=1.6U w=12.0U
MN5  out  i2 gnd gnd      NMOS l=1.6U w=12.0U
MN6  out  i3 gnd gnd      NMOS l=1.6U w=12.0U
.ends nor3

.subckt nor4 i1 i2 i3 i4 out vcc
MP1  _2   i1 vcc vcc      PMOS l=1.6U w=12.0U
MP2  _3   i2 _2  vcc      PMOS l=1.6U w=12.0U
MP3  _4   i3 _3  vcc      PMOS l=1.6U w=12.0U
MP4  out  i4 _4  vcc      PMOS l=1.6U w=12.0U
MN5  out  i1 gnd gnd      NMOS l=1.6U w=12.0U
MN6  out  i2 gnd gnd      NMOS l=1.6U w=12.0U
MN7  out  i3 gnd gnd      NMOS l=1.6U w=12.0U
MN8  out  i4 gnd gnd      NMOS l=1.6U w=12.0U
.ends nor4

.subckt or2 i1 i2 out vcc
X1 i1 i2 _1               vcc    nor2
X2 _1 out                 vcc    inv
.ends or2

.subckt or3 i1 i2 i3 out vcc
X1 i1 i2 i3 _1            vcc    nor3
X2 _1 out                 vcc    inv
.ends or3

.subckt or4 i1 i2 i3 i4 out vcc
X1 i1 i2 i3 i4 _1         vcc    nor4
X2 _1 out                 vcc    inv
.ends or4

.subckt dffG D  q  ck vcc
X1 _4 _2 _1       vcc nand2
X2 ck _1 _2       vcc nand2
X3 ck _2 _4 _3    vcc nand3
X4 D  _3 _4       vcc nand2
X5 _3 q  nq       vcc nand2
X6 _2 nq q        vcc nand2
.ends dffG

.subckt dff D  q  clk vcc
MP1  nh   clk vcc vcc     PMOS l=1.6U w=12.0U
MN2  nh   clk gnd gnd     NMOS l=1.6U w=12.0U
MP3  h    nh  vcc vcc     PMOS l=1.6U w=12.0U
MN4  h    nh  gnd gnd     NMOS l=1.6U w=12.0U
MP5  _2   _1  vcc vcc     PMOS l=1.6U w=12.0U
MN6  _2   _1  gnd gnd     NMOS l=1.6U w=12.0U
MP7  _3   _2  vcc vcc     PMOS l=1.6U w=12.0U
MN8  _3   _2  gnd gnd     NMOS l=1.6U w=12.0U
MP9  _5   _4  vcc vcc     PMOS l=1.6U w=12.0U
MN10 _5   _4  gnd gnd     NMOS l=1.6U w=12.0U
MP11 q    _5  vcc vcc     PMOS l=1.6U w=12.0U
MN12 q    _5  gnd gnd     NMOS l=1.6U w=12.0U
MN13 D    h   _1  gnd     NMOS l=1.6U w=12.0U
MP14 D    nh  _1  vcc     PMOS l=1.6U w=12.0U
MN15 _1   nh  _3  gnd     NMOS l=1.6U w=12.0U
MP16 _1   h   _3  vcc     PMOS l=1.6U w=12.0U
MN17 _3   nh  _4  gnd     NMOS l=1.6U w=12.0U
MP18 _3   h   _4  vcc     PMOS l=1.6U w=12.0U
MN19 _4   h   q   gnd     NMOS l=1.6U w=12.0U
MP20 _4   nh  q   vcc     PMOS l=1.6U w=12.0U
.ends dff
```

```
.subckt dffclear D      CLRZ clk q nq vcc
M01  nh   clk    vcc  vcc  PMOS l=1.6U w=12.0U
M02  nh   clk    gnd  gnd  NMOS l=1.6U w=12.0U
M03  h    nh     vcc  vcc  PMOS l=1.6U w=12.0U
M04  h    nh     gnd  gnd  NMOS l=1.6U w=12.0U
M05  _2   CLRZ   vcc  vcc  PMOS l=1.6U w=12.0U
M06  _2   _1     vcc  vcc  PMOS l=1.6U w=12.0U
M07  _2   CLRZ   _7   gnd  NMOS l=1.6U w=12.0U
M08  _7   _1     gnd  gnd  NMOS l=1.6U w=12.0U
M09  _3   _2     gnd  gnd  NMOS l=1.6U w=12.0U
M10  _3   _2     vcc  vcc  PMOS l=1.6U w=12.0U
M11  nq   CLRZ   vcc  vcc  PMOS l=1.6U w=12.0U
M12  nq   _4     vcc  vcc  PMOS l=1.6U w=12.0U
M13  nq   CLRZ   _8   gnd  NMOS l=1.6U w=12.0U
M14  _8   _4     gnd  gnd  NMOS l=1.6U w=12.0U
M15  q    nq     gnd  gnd  NMOS l=1.6U w=12.0U
M16  q    nq     vcc  vcc  PMOS l=1.6U w=12.0U
M17  D    h      _1   gnd  NMOS l=1.6U w=12.0U
M18  D    nh     _1   vcc  PMOS l=1.6U w=12.0U
M19  _1   nh     _3   gnd  NMOS l=1.6U w=12.0U
M20  _1   h      _3   vcc  PMOS l=1.6U w=12.0U
M21  _3   nh     _4   gnd  NMOS l=1.6U w=12.0U
M22  _3   h      _4   vcc  PMOS l=1.6U w=12.0U
M23  _4   h      q    gnd  NMOS l=1.6U w=12.0U
M24  _4   nh     q    vcc  PMOS l=1.6U w=12.0U
.ends  dffclear

.subckt add1 a b  carry sum vcc
MP1  _3   a  vcc vcc  PMOS l=1.6U w=12.0U
MP2  ca   b  _3  vcc  PMOS l=1.6U w=12.0U
MN3  ca   a  gnd gnd  NMOS l=1.6U w=12.0U
MN4ca     b  gnd gnd  NMOS l=1.6U w=12.0U
MP5  _5   a  vcc vcc  PMOS l=1.6U w=12.0U
MP6  _5   b  vcc vcc  PMOS l=1.6U w=12.0U
MP7  s0   ca _5  vcc  PMOS l=1.6U w=12.0U
MN8  s0   ca gnd gnd  NMOS l=1.6U w=12.0U
MN9  _6   a  gnd gnd  NMOS l=1.6U w=12.0U
MN10 s0   b  _6  gnd  NMOS l=1.6U w=12.0U
X1 s0 sum        vcc inv
X2 ca carry      vcc inv
.ends add1

.subckt demi-add a b  carry sum vcc
MP1  _3  a  vcc vcc      PMOS l=1.6U w=12.0U
MP2  so  b  _3  vcc      PMOS l=1.6U w=12.0U
MP3  so  ca vcc vcc      PMOS l=1.6U w=12.0U
MP4  ca  a  vcc vcc      PMOS l=1.6U w=12.0U
MP5  ca  b  vcc vcc      PMOS l=1.6U w=12.0U
MN6  _2  ca so  gnd      NMOS l=1.6U w=12.0U
MN7  _2  a  gnd gnd      NMOS l=1.6U w=12.0U
MN8  _2  b  gnd gnd      NMOS l=1.6U w=12.0U
MN9  ca  a  _5  gnd      NMOS l=1.6U w=12.0U
MN10 _5  b  gnd gnd      NMOS l=1.6U w=12.0U
X1 so sum        vcc inv
X2 ca carry vcc inv
.ends demi-add


.subckt adder a b c carry sum vcc
M1 carry  _16   vcc  vcc  PMOS l=1.6U w=12.0U
M2 _15    a     vcc vcc   PMOS l=1.6U w=12.0U
M3 vcc    b     _15 vcc   PMOS l=1.6U w=12.0U
M4 _15    c     _16 vcc   PMOS l=1.6U w=12.0U
M5 _16    b     _9  vcc   PMOS l=1.6U w=12.0U
M6 _9     a     _15 vcc   PMOS l=1.6U w=12.0U
M7 _7     a     _5  vcc   PMOS l=1.6U w=12.0U
M8 _5     b     _3  vcc   PMOS l=1.6U w=12.0U
M9 _3     c     _8  vcc   PMOS l=1.6U w=12.0U
M10 _8    _16   _7  vcc   PMOS l=1.6U w=12.0U
M11 _7    c     vcc vcc   PMOS l=1.6U w=12.0U
M12 vcc   b     _7  vcc   PMOS l=1.6U w=12.0U
M13 _7    a     vcc vcc   PMOS l=1.6U w=12.0U
M14 vcc   _8    sum vcc   PMOS l=1.6U w=12.0U
M15 carry _16   gnd gnd   NMOS l=1.6U w=12.0U
M16 gnd   a     _13 gnd   NMOS l=1.6U w=12.0U
M17 _13   b     _16 gnd   NMOS l=1.6U w=12.0U
M18 _16   c     _10 gnd   NMOS l=1.6U w=12.0U
M19 _10   b     gnd gnd   NMOS l=1.6U w=12.0U
M20 gnd   a     _10 gnd   NMOS l=1.6U w=12.0U
M21 gnd   a     _6  gnd   NMOS l=1.6U w=12.0U
M22 _6    b     _4  gnd   NMOS l=1.6U w=12.0U
M23 _4    c     _8  gnd   NMOS l=1.6U w=12.0U
M24 _8    _16   _2  gnd   NMOS l=1.6U w=12.0U
M25 _2    c     gnd gnd   NMOS l=1.6U w=12.0U
M26 gnd   b     _2  gnd   NMOS l=1.6U w=12.0U
M27 _2    a     gnd gnd   NMOS l=1.6U w=12.0U
M28 gnd   _8    sum gnd   NMOS l=1.6U w=12.0U
.ends adder

.subckt nxor_nand a b prop y vcc
MP1  _3   a     vcc  vcc  PMOS l=1.6U w=12.0U
MP2  prop b     _3   vcc  PMOS l=1.6U w=12.0U
MP3  prop y     vcc  vcc  PMOS l=1.6U w=12.0U
MN4  _4   y     prop gnd  NMOS l=1.6U w=12.0U
MN5  _4   a     gnd  gnd  NMOS l=1.6U w=12.0U
MN6  _4   b     gnd  gnd  NMOS l=1.6U w=12.0U
MP7  y    a     vcc  vcc  PMOS l=1.6U w=12.0U
MP8  y    b     vcc  vcc  PMOS l=1.6U w=12.0U
MN9  y    a     _5   gnd  NMOS l=1.6U w=12.0U
MN10 _5   b     gnd  gnd  NMOS l=1.6U w=12.0U
.ends nxor_nand

.subckt cla1 ng0 np0 cin out vcc
M1 out ng0  vcc  vcc     PMOS  l=1.6U w=12U
M2 _1  np0  vcc  vcc     PMOS  l=1.6U w=12U
M3 out cin  _1   vcc     PMOS  l=1.6U w=12U
M4 out ng0  _2   gnd     NMOS  l=1.6U w=12U
M5 _2  np0  gnd  gnd     NMOS  l=1.6U w=12U
M6 _2  cin  gnd  gnd     NMOS  l=1.6U w=12U
.ends cla1

.subckt cla2 ng0 np0 ng1 np1 cin out vcc
M1  out ng1  vcc vcc     PMOS  l=1.6U w=12U
M2  _2 ng0   vcc vcc     PMOS  l=1.6U w=12U
M3  _1 np0   vcc vcc     PMOS  l=1.6U w=12U
M4  _2 cin   _1  vcc     PMOS  l=1.6U w=12U
M5  out np1  _2  vcc     PMOS  l=1.6U w=12U
M6  out ng1  _3  gnd     NMOS  l=1.6U w=12U
M7  _3  np1  gnd gnd     NMOS  l=1.6U w=12U
M8  _3  np0  _4  gnd     NMOS  l=1.6U w=12U
M9  _3  cin  _4  gnd     NMOS  l=1.6U w=12U
M10 _4  ng0  gnd gnd     NMOS  l=1.6U w=12U
.ends cla2
```

```
.subckt cla3 ng0 np0 ng1 np1 ng2 np2 cin out vcc
M1  out ng2  vcc vcc     PMOS  l=1.6U w=12U
M2  _1  np2  vcc vcc     PMOS  l=1.6U w=12U
M3  out ng1  _1  vcc     PMOS  l=1.6U w=12U
M4  _2  ng0  _1  vcc     PMOS  l=1.6U w=12U
M5  _3  np0  _1  vcc     PMOS  l=1.6U w=12U
M6  _2  cin  _3  vcc     PMOS  l=1.6U w=12U
M7  out np1  _2  vcc     PMOS  l=1.6U w=12U
M8  out ng2  _4  gnd     NMOS  l=1.6U w=12U
M9  _4  np2  gnd gnd     NMOS  l=1.6U w=12U
M10 _4  np1  _5  gnd     NMOS  l=1.6U w=12U
M11 _4  ng0  _6  gnd     NMOS  l=1.6U w=12U
M12 _6  np0  _5  gnd     NMOS  l=1.6U w=12U
M13 _6  cin  _5  gnd     NMOS  l=1.6U w=12U
M14 _5  ng1  gnd gnd     NMOS  l=1.6U w=12U
.ends cla3

.subckt cla4 ng0 np0 ng1 np1 ng2 np2 ng3 np3
+            cin out vcc
M1  _4  ng2  vcc vcc     PMOS  l=1.6U w=12U
M2  _1  np2  vcc vcc     PMOS  l=1.6U w=12U
M3  _4  ng1  _1  vcc     PMOS  l=1.6U w=12U
M4  _2  ng0  _1  vcc     PMOS  l=1.6U w=12U
M5  _3  np0  _1  vcc     PMOS  l=1.6U w=12U
M6  _2  cin  _3  vcc     PMOS  l=1.6U w=12U
M7  _4  np1  _2  vcc     PMOS  l=1.6U w=12U
M8  out np3  _4  vcc     PMOS  l=1.6U w=12U
M9  out ng3  vcc vcc     PMOS  l=1.6U w=12U
M10 out ng3  _5  gnd     NMOS  l=1.6U w=12U
M11 _5  np3  gnd gnd     NMOS  l=1.6U w=12U
M12 _5  np2  _7  gnd     NMOS  l=1.6U w=12U
M13 _7  ng2  gnd gnd     NMOS  l=1.6U w=12U
M14 _5  ng1  _6  gnd     NMOS  l=1.6U w=12U
M15 _6  np1  _7  gnd     NMOS  l=1.6U w=12U
M16 _6  np0  _8  gnd     NMOS  l=1.6U w=12U
M17 _6  cin  _8  gnd     NMOS  l=1.6U w=12U
M18 _8  ng0  _7  gnd     NMOS  l=1.6U w=12U
.ends cla4
```

```
.subckt adder4    cin a0 a1 a2 a3 b0 b1 b2 b3 s0 s1
+                 s2 s3 cout vcc
X1  a0 b0 np0 ng0                    vcc nxor_nand
X2  a1 b1 np1 ng1                    vcc nxor_nand
X3  a2 b2 np2 ng2                    vcc nxor_nand
X4  a3 b3 np3 ng3                    vcc nxor_nand
X5  cin ncin                         vcc inv
X6  ng0 np0 ncin c1                  vcc cla1
X7  ng0 np0 ng1 np1 ncin c2          vcc cla2
X8  ng0 np0 ng1 np1 ng2 np2          ncin c3  vcc cla3
X9  ng0 np0 ng1 np1 ng2 np2 ng3 np3 ncin cout vcc cla4
X10 cin np0 s0 k0                    vcc nxor_nand
X11 c1  np1 s1 k1                    vcc nxor_nand
X12 c2  np2 s2 k2                    vcc nxor_nand
X13 c3  np3 s3 k3                    vcc nxor_nand
.ends adder4

.subckt adder4B   ncin a0 a1 a2 a3 b0 b1 b2 b3 s0 s1
+                 s2 s3 ncout vcc
X1  a0 b0 np0 ng0                    vcc nxor_nand
X2  a1 b1 np1 ng1                    vcc nxor_nand
X3  a2 b2 np2 ng2                    vcc nxor_nand
X4  a3 b3 np3 ng3                    vcc nxor_nand
X5  ng0 np0      ncin c1             vcc cla1
X6  ng0 np0 ng1 np1      ncin c2     vcc cla2
X7  ng0 np0 ng1 np1 ng2 np2          ncin c3  vcc cla3
X8  ng0 np0 ng1 np1 ng2 np2 ng3 np3 ncin cout vcc cla4
X9  cout ncout                       vcc buf
X10 ncin np0 s0                      vcc xor
X11 c1   np1 s1                      vcc nxor
X12 c2   np2 s2                      vcc nxor
x13 c3   np3 s3                      vcc nxor
.ends adder4B

.subckt xorgate i1 i2 out vcc
X1  i1 i2 _1      vcc nand2
X1  i1 _1 _2      vcc nand2
X1  i2 _1 _3      vcc nand2
X1  _2 _3 out     vcc nand2
.ends xorgate

*** END LIBRAIRIE
```

ANNEXE 3

Conversion de la description symbolique vers la base de données de CADENCE

(1) Conversion de la description symbolique (format PRINT) - EDGE

```
/*****************************************************************************
  take the print description and generates the symbolic cadence description
*****************************************************************************/
(setq LstProps
    (list
        '("graphicsEditorUnits per userUnit" "float" 40.0)
        '("screenGridSpacing" "int" 10)
        '("snapSpacing" "int" 10)
        '("sySpacingRules" "string" "techfile")
        '("syPreservePathWidth" "Boolean" t)
        '("syAutoJog" "Boolean" t)
        '("syMaxIterations" "int" 2)
    )
)

(setq LstLayers
    (list
        '(17 "DIFFUSIONN")
        '(18 "DIFFUSIONP")
        '(25 "POLYSILICIUM")
        '(40 "METAL1")
        '(42 "METAL2")
    )
)

(setq LstDevices
    (list
        ; CadName      PName          PType
        '("TransP"     "TRANSISTOR"   "MOSP")
        '("TransN"     "TRANSISTOR"   "MOSN")
        '("Cpm1"       "CONTACT"      "POLYMET1")
        '("Cdnm1"      "CONTACT"      "DIFFNMET1")
        '("Cdpm1"      "CONTACT"      "DIFFPMET1")
        '("Cm1m2"      "CONTACT"      "MET1MET2")
        '("Cwel"       "CONTACT"      "BODY-N+")
        '("Csub"       "CONTACT"      "BODY-P+")
    )
)
```

```
/*************************************************************************/
procedure( recognize(syl)
prog( (c)
 c = getc(syl)
 while( (isseparateur(c)==1)  c=getc(syl) )
 if( (c=='D)
     case( getc(syl)
           ( 'S  return(1))
           ( 'P  return(2))
           ( 'L  return(3))
           ( t   return(-1))
          )
   )
 if( (c=='I)
     case( getc(syl)
           ( 'S  return(4))
           ( 'P  return(5))
           ( 'L  return(6))
           ( t   return(-1))
          )
   )
 if( (c=='A)  return(7))
 if( (c=='B)  return(8))
 if( (c=='N)  return(9))
 if( (c=='P)  return(10))
 if( (c=='C)  return(11))
 return(-1)
 )
)

/*************************************************************************/
procedure( isseparateur(car)
prog( ()
 if( car=='\ || car=='\n || car=='\t || car=='\r || car=='\/ || car=='\* || car=='\= || car=='\,   return(1) )
 if( car==nil || car=='\; return(-1))
 return(0)
 )
)

/*************************************************************************/
; observation: la variable 'token' est  global
procedure( search_word(syl)
 prog( (c)
 token = ""
 c = getc(syl)
 while( (isseparateur(c)==1)  c=getc(syl) )
 while( (isseparateur(c)==0)
          token = strcat(token c)
          c = getc(syl)
     )
 return(c)
 )
)

/*************************************************************************/
procedure( inst_symbol(syl rep)
prog( (nombre x y mst inst)

   fscanf(syl "%d %d %d" nombre x y)
   foreach(k l_def
     if(nth(0 k)==nombre then
      ;printf("S %s %f %f %d %d\n" nth(1 k) nth(2 k) nth(3 k) x y )
       if( nth(1 k)=="TransP" || nth(1 k)=="TransN" then
```

```
        mst = dbOpen( list(nth(1 k) "symbolic") )
        inst = dbCreateInst(rep mst nth(1 k) (x*snap):(y*snap) "90")
        dbReplaceProp(inst "w" "float" nth(2 k))
        dbReplaceProp(inst "l" "float" nth(3 k))
        dbClose(mst)
       else
        mst = dbOpen( list(nth(1 k) "symbolic") )
       inst = dbCreateInst(rep mst nth(1 k) (x*snap):(y*snap) "0")
        dbReplaceProp(inst "w"      "float" contW)
        dbReplaceProp(inst "l"      "float" contW)
        dbReplaceProp(inst "xPitch" "float" contH)
        dbReplaceProp(inst "yPitch" "float" contH)
        dbReplaceProp(inst "column" "int" fix(nth(2 k)/contH))
        dbReplaceProp(inst "row"    "int" fix(nth(3 k)/contH))
        dbClose(mst)
       )
      )
        )

  c = search_word(syl)  ; look for labeled signals
  (if c!=\; then
     (while getc(syl)!=\; nil)      ; take the end of the line
    net = dbFindNetByName(rep token) ; take the net associated to the name
    if( net==nil net=dbCreateNet(rep token))
    pin = dbCreatePin(net inst token) ; put a pin under the contact
    ; printf("signal= %s \n" token )
  )
 )
)

/*************************************************************************/
procedure( inst_pin(syl rep)
prog( (nombre x y mst inst)

  fscanf(syl "%d %d %d" nombre x y)
  search_word(syl)
  ;printf("P %s %d %d\n" token x y)
  if( token=="vdd!" || token=="gnd!" then
      mst = dbOpen("m1_t symbolic")
      inst = dbCreateInst(rep mst token (x*snap):(y*snap) "0")
          dbClose(mst)
     else
      mst = dbOpen("m2_t symbolic")
      inst = dbCreateInst(rep mst token (x*snap):(y*snap) "0")
          dbClose(mst)
    )

  net = dbFindNetByName(rep token)
  if( net==nil net = dbCreateNet(rep token))
  pin = dbCreatePin(net inst nil)
  while( getc(syl)!=\; nil)  ; prendre le fin de la ligne
 )
)

/*************************************************************************/
procedure( inst_wire(syl rep)
prog( (nombre width x y x2 y2 i)

  declare(a[3])
  fscanf(syl "%d %d %d" nombre x y)

  c=\
  i=0
```

```
  while( c!='\;  c=search_word(syl)
                 a[i] = evalstring(token)
                    i = i + 1
      )
  ; optional wire width : x1 y1 /width/ x2 y2
  if( (i==2)  then x2=a[0]  y2=a[1]  width=0.0
          else x2=a[1]  y2=a[2]  width=a[0]
    )

  ; printf("L  [%f] %d %d %d %d\n" width x y x2 y2)

  foreach(k l_wire
        if(nth(0 k)==nombre then
           if( (width==0.0)  width=nth(2 k))
           dbCreatePath(rep nth(1 k) list((x*snap):(y*snap) (x2*snap):(y2*snap)) width)
        )
        )
 )
)

/*****************************************MAIN  PROCEDURE  ****************/
procedure( sread(output @optional snap)
 prog( (n syl rep input)

  l_wire=l_def=rep=nil
  n=0
  (if (snap==nil) then snap=10)

  dbSetPath(". techfile.dev")
  sprintf(input "%s.syl" output)
  syl = infile(input)
  if( syl==nil then printf("\nfichier introuvable\n") return(1))


  printf("\nInput  file: %s\n" input)
  printf("Output file: %s\n" output)
  printf("Snap:        %d\n" snap)

  rep = dbOpen(list(output "symbolic") nil "a")
  ; efface la representation anterieur
  dbDeleteAllNet(rep)
  foreach(inst rep~>instances dbDeleteObject(inst))
  foreach(inst rep~>terminals dbDeleteObject(inst))
  foreach(inst rep~>shapes    dbDeleteObject(inst))
  foreach(inst rep~>prop      dbDeletePropByName(rep inst~>name))

  foreach(k LstProps dbReplaceProp(rep nth(0 k) nth(1 k) nth(2 k)))

  while( n!=-1
   n=recognize(syl)
   case( n
         (1 fscanf(syl "%d" nombre)
           while( search_word(syl)!='\;
             case(token
                   ("Type" search_word(syl) type=token )
                   (("W2" "W") fscanf(syl "%f" w) )
                   (("L2" "L") fscanf(syl "%f" l) ))
           )
           foreach(k LstDevices  if( nth(2 k)==type  symb=nth(0 k)))
           l_def = cons( list(nombre symb w l) l_def)
         )
         (2 while( getc(syl)!='\; nil))
         (3 fscanf(syl "%d" nombre)
```

```
            while( search_word(syl)!=\;
              case(token
                    ("Name"    search_word(syl) type=token )
                    ("Largeur"  fscanf(syl "%f" w) )
                    )
              )
            foreach(k LstLayers  if( nth(1 k)==type  level=nth(0,k)))
            l_wire = cons( list(nombre level w) l_wire)
          )
          (4 inst_symbol(syl rep))
          (5 inst_pin(syl rep))
          (6 inst_wire(syl rep))
          (7 fscanf(syl "%d %d %d %d" x y x2 y2)
            ;printf("bBox %d %d %d %d\n" x y x2 y2)
            inst = dbCreateRect(rep list("prboundary" "boundary")
                        list((x*snap):(y*snap) (x2*snap):(y2*snap)))
        dbReplaceProp(inst "syCellBoundary" "Boolean" t)
        dbReplaceProp(inst "syStretchable" "Boolean" t)
            while( getc(syl)!=\; nil)
          )
        (8 fscanf(syl "%s %d %d %d %d" ss x y x2 y2)
          dbCreateRect(rep "softFence" list((x*snap):(y*snap-snap/2)
                              (x2*snap):(y2*snap+snap/2)))
            while( getc(syl)!=\; nil)
          )
        (9 fscanf(syl "%s %d %d %d %d" ss x y x2 y2)
          dbCreateRect(rep "CNWI" list(((x-2)*snap):(y*snap-snap)
                              ((x2+2)*snap):(y2*snap+snap)))
            while( getc(syl)!=\; nil)
        )
        (10 fscanf(syl "%s %d %d %d %d" ss x y x2 y2)
          ; pwell isn't used
            while( getc(syl)!=\; nil)
        )
        (11 fscanf(syl "%s %f %f" ss contW contH)
            while( getc(syl)!=\; nil)
          )
      )
    )
  dbSave(rep)
  dbClose(rep)
  return(t)
  )
)
/*****************************************************************************/
```

(2) Conversion de la description symbolique - OPUS

Remarque : nous avons adopté une langage symbolique plus simple que celle définie par PRINT (option "-c" dans l'appel de TROPIC). La syntaxe est montrée au début du convertisseur.

```
/*********************************************************************
SINTAXE:
        Transistor :    T <type> x y w l
                            <type> = TransN, TransP
        Contact    :    C <type> x y nw nl text
                            <type> = Cpm1, Cdnm1, Cdpm1, Cm1m2, Cwel, Csub
                            nw = how many contacts in Y
                            nl = how many contacts in X
                            text = contact name or "nil"
        Wire (line):    L <type> x1 y1 x2 y2 width
                            <type> = metal1, metal2 , poly, diffN, diffP
        Pin :           P <type> x y name width
                            <type> = m1_t, m2_t
        Alignement :    F x1 y1 x2 y2
        Well :          W <type> x1 y1 x2 y2
                            <type> = NWELL, PWELL
        Boundary   : x1 y1 x2 y2
*********************************************************************/

DEFAULT_LIB = "ISCAS89"

procedure( conv(output \@optional align snap)
  prog( (lib instTN instTP instP1 inst P2 inst tr rep c syl type x1 x2 y1 y2 w l txt ct_txt ok wr w1 w2 contW
pitch)

    ok = 1
    ct_txt = 0
    if( (snap == nil) then (snap = 10))
    if( (align == nil)
       then printf("******* without transistor alignement") else printf("******* transistor alignement") )
    /* get the current library name and the list of the wires */
    lib = dmGetCurLib()
    if(lib==nil then lib = dmOpenLib(DEFAULT_LIB "" nil "a"))

    sprintf(txt "%s.syl" output)                    /* open the symbolic file description */
    syl = infile(txt)
    if( (syl == nil) then printf("\nfichier introuvable\n") return(1))
    dmDeleteCell(lib output)      /* kill the old symbolic file */

     /* set the symbolic parameters */
    rep = dbOpenCellView(lib output "symbolic" nil "a") sySetAutoJog(rep t)
    sySetMaxAutoJogs(rep 6)
    sySetMaxIterations(rep 2.0)
    sySetPreserveWireWidth(rep t)

     /* get the contact size and the transistors pointers*/
    inst = dbOpenCellView(lib "Cpm1" "symbolic" nil "r")
    contW = inst~>parameters~>w
    pitch = inst~>parameters~>xPitch
    dbClose(inst)
    instTN = dbOpenCellView(lib "TransN" "symbolic" nil "r")
    instTP = dbOpenCellView(lib "TransP" "symbolic" nil "r")
    instP1 = dbOpenCellView(lib "m1_t" "symbolic" nil "r")
    instP2 = dbOpenCellView(lib "m2_t" "symbolic" nil "r")
```

```
while( (ok == 1)
        c = getc(syl)
        case( c
            ('T
                fscanf(syl "%s %d %d %f %f\n" type x1 y1 w l)
                sprintf(txt "tr%d" ct_txt)
                ct_txt++
                if( (type=="TransN")  then (inst = instTN) else if( (type == "TransP") then (inst = instTP)))
                tr = dbCreateInst(rep inst txt ((x1 * snap):(y1 * snap)) "R90")
                dbReplaceProp(tr "w" "float" w)
                dbReplaceProp(tr "l" "float" l))
            ('C
                fscanf(syl "%s %d %d %d %d %s\n" type x1 y1 w l name)
                leCreateContact(rep type ((x1 * snap):(y1 * snap)) "R0" contW contW  w l pitch pitch "center" "center")
                if( (name != "nil")
                        dbCreateLabel(rep "text" ((x1 * snap):(y1 * snap)) name "centerCenter" "R0" "roman" 1)))
            ('P
                fscanf(syl "%s %d %d %s %f\n" type x1 y1 name w) sprintf(txt "tr%d" ct_txt)
                ct_txt++
                if( (type == "m2_t")  then (inst = instP2) else if( (type == "m1_t") then (inst = instP1)))
                tr = dbCreateInst(rep inst txt ((x1 * snap):(y1 * snap)) "R0")
                dbReplaceProp(tr "w" "float" w)
                dbReplaceProp(tr "l" "float" w)
                wr = dbMakeNet(rep name)
                dbCreatePin(wr tr name)
                dbCreateLabel(rep "text" ((x1*snap):(y1*snap)) name "centerCenter" "R0" "roman" 1))
            ('L
                fscanf(syl "%s %d %d %d %d %f\n" type x1 y1 x2 y2 w)
                wr = tcGetWire(type lib)
                w1 = dbCreatePath(rep nth(0 wr) list(((x1*snap):(y1*snap)) ((x2*snap):(y2*snap))) w)
                if( (nth(4 wr) != nil)                    /* N or P implant */
                        then w2 = dbCreatePath(rep nth(0 nth(0 nth(4 wr))) list(((x1*snap):(y1*snap))
                                        ((x2*snap):(y2*snap))) ((2*nth(1 nth(0 nth(4 wr))))+w))
                    dbSetq(w2 w1 parent) /* means : w2 is attached to w1 */ dbSetq(w1 t matchPoints)))
            ('F
                fscanf(syl "%d %d %d %d\n" x1 y1 x2 y2)
                if( (align!=nil) then  dbCreateRect(rep "softFence" list((x1*snap):(y1*snap-1) (x2*snap):(y2*snap)))))
            ('B
                fscanf(syl "%d %d %d %d\n" x1 y1 x2 y2)
                inst = dbCreateRect(rep list("prBoundary" "boundary") list((x1*snap):(y1*snap) (x2*snap):(y2*snap)))
                dbReplaceProp(inst "syCellBoundary" "Boolean" t)
                dbReplaceProp(inst "syStretchable" "Boolean" t))
            ('W
                fscanf(syl "%s %d %d %d %d\n" type x1 y1 x2 y2)
                if( (type=="NWELL") then
                    dbCreateRect(rep "CNWI" list((x1*snap):(y1*snap-snap) (x2*snap):(y2*snap+snap)))))
            (t  ok=0 )
        )
    )

    dbSave(rep)    dbClose(instTN)    dbClose(instTP)    dbClose(instP1)    dbClose(instP2)

    /* call the compaction procedure, an transform it to polygonon */
    inst = syCompactView(rep list(output "compacted") nil)
    dbReplaceProp(inst "syConvertMode" "string" "fill gap")
    dbReplaceProp(inst "sySmashDevice" "boolean" t)
    instTN = sySymToPolygon(inst list(output "layout"))
    dbSave(instTN)                dbClose(instTN)
    dmDeleteCellView(lib output "compacted")
    dbClose(rep)                  dmCloseLib(lib)
    return(t)
)
)
```

ANNEXE 4

Description des règles de dessin

(1) Format EDGE - technologie ECPD07

```
! fichier technologique techno ecpd07.
! version utilisant ndiff et pdiff
! Moraes le 02-11-93.

*SYSTEM
   UNIT              : UM          ! Specifies user unit value
   DB_SCALE          : 0.025       ! Specifies database unit value
   GRID_RESOLUTION : 0.1           ! Specifies minimum digitizing grid
*END

*LAYER
  CNWI          10    WELL          CNWI          ORANGE
  CTOX          15    DIFFUSION     CTOX          GREEN
  ndiff         17    DIFFUSION     CTOX          GREEN
  pdiff         18    DIFFUSION     CTOX          GREEN
  CNPI          30    IMPLANT       CNPI          GREEN
  CPPI          32    IMPLANT       CPPI          YELLOW
  CPOL          25    CONDUCTION    CPOL          RED
  CCON          35    VIA_CUT       CCON          MAGENTA
  CVIA          37    VIA_CUT       CVIA          RED
  CME1          40    CONDUCTION    CME1          BLUE
  CME2          42    CONDUCTION    CME2          CYAN
  CPAS          45    PADOPENING    CPAS          CYAN
  softFence     230   SOFT_FENCE    softFence     MAGENTA
  hardFence     232   HARD_FENCE    hardFence     RED
  prboundary    130   BOUNDARY      prboundary    GREEN
*END

*DEVICE

  ENHANCEMENT:
!   Trans      Diff     Gate    Gate     Gate     S/D     Gate
!   Name       Layer    Layer   Width    Length   Width   Ext.
!   ------------------------------------------------------------
    TransP     pdiff    CPOL    1.0      0.8      1.2     0.8
    TransN     ndiff    CPOL    1.0      0.8      1.2     0.8

  CONTACT:
!   Cnt        Cnt      Cndctr  Cndctr   Cnt      Cnt     Enc     Enc
!   Name       Layer    Layer1  Layer2   Wdth     Hght    Lay1    Lay2
!   ------------------------------------------------------------------
    Cpm1       CCON     CME1    CPOL     1.0      1.0     0.5     0.5
    Cdnm1      CCON     CME1    ndiff    1.0      1.0     0.5     0.5
    Cdpm1      CCON     CME1    pdiff    1.0      1.0     0.5     0.5
    Cm1m2      CVIA     CME1    CME2     1.0      1.0     0.5     0.5
```

SUBSTRATE_CONTACT:

! Cnt Name	Cnt Layer	Diff Layer	Cndctr Layer	Cnt Wdth	Cnt Hght	Enc Diff.	Enc Cond.
Cwel	CCON	ndiff	CME1	1.0	1.0	0.5	0.5
Csub	CCON	pdiff	CME1	1.0	1.0	0.5	0.5

PIN:

! Pin Name	Layer Name	Pin Width
m1_t	CME1	1.2
m2_t	CME2	1.2
polyt	CPOL	0.8

WIRE:

! Wire Name	Wire Layer	Wire Width	Legal Region	Wire Resist.
CPOL	CPOL	0.8		33
CME1	CME1	1.2		0.06
CME2	CME2	1.2		0.03
ndiff	ndiff	0.8		40
pdiff	pdiff	0.8		80

*END

*DESIGN_RULE

! DRC Command	Edge Layer1	Edge Layer2	Relational Operator	DRC Value	commentaire (ES2 rule number)
EXT	CNWI	CNWI	GE	4.0	! 102
WIDTH	CNWI		GE	4.0	! 101
WIDTH	CTOX		GE	0.8	! 201
EXT	CTOX	CTOX	GE	1.6	! 202
ENC	CTOX	CNWI	GE	2.4	! 203
ENC	CTOX(Cwel)	CNWI	GE	0.0	! 204
EXT	CTOX	CNWI	GE	2.4	! 205 207
WIDTH	CPOL		GE	0.8	! 501
EXT	CPOL	CPOL	GE	1.2	! 502 504
EXT	CPOL	CTOX	GE	0.4	! 506 507
EXT	CCON	CCON	GE	1.0	! 702
EXT	CCON(Cpm1)	CTOX	GE	0.7	! 706
EXT	CCON(Cdpm1)	CPOL	GE	0.7	! 707
EXT	CCON(Cdnm1)	CPOL	GE	0.7	! 707
WIDTH	CME1		GE	1.2	! 801
EXT	CME1	CME1	GE	1.2	! 802
EXT	CVIA	CVIA	GE	1.0	! 754
EXT	CVIA	CPOL	GE	1.1	! 757
EXT	CVIA	CCON	GE	1.0	! 759
WIDTH	CME2		GE	1.2	! 851
EXT	CME2	CME2	GE	1.2	! 852
EXT	CNPI	CPPI	GE	0.0	! 661

*END

(2) Format OPUS - technologie ECPD10

Contient les règles de dessin et les règles pour LAS.

```
; Cadence Technology File
; pour symbolique ecpd10

tfcDefineView(
; (ViewName      ViewType)
  (layout        maskLayout)
  (compacted     maskLayout)
  (symbolic      maskLayout)
  (las           maskLayout)
)

tfcDefineViewPropByViewType(
; (ViewType PropName              PropValue)
  (maskLayout startLevel          0)
  (maskLayout stopLevel           5)
  (maskLayout userUnits           "micron")
  (maskLayout DBUPerUU            1000.0)
  (maskLayout drawGridOn          nil)
  (maskLayout gridSpacing         125)
  (maskLayout gridMultiple        8)
  (maskLayout xSnapSpacing        .125)
  (maskLayout ySnapSpacing        125)
  (maskLayout minGridResolution .001)
)

tfcDefineViewProp(
; (Viewname PropName              PropValue)
  (symbolic gridSpacing           5.0)
  (symbolic gridMultiple          5)
  (symbolic xSnapSpacing          5.0)
  (symbolic ySnapSpacing          5.0)

  (symbolic lasPrimitiveList      "mos PMOS pmos NMOS nmos pfet nfet" )
  (symbolic lasPseudoElementList "vdd gnd" )
  (symbolic lasBulkNameList       "B b" )
  (symbolic lasGateNameList       "G g" )
  (symbolic lasSourceNameList     "S s" )
  (symbolic lasDrainNameList      "D d" )
  (symbolic lasSwitchViewList     "schematic cmos.sch netlist" )
  (symbolic lasLengthName         "l" )
  (symbolic lasWidthName          "w" )
  (symbolic lasVddNameList        "vdd! VCC VDD PWR vcc vdd")
  (symbolic lasVssNameList        "gnd! VSS GND vss gnd")
  (symbolic lasInstTypePropName "subtype" )
  (symbolic lasMasterTypePropName "instNamePrefix" )
)

; Layer Section
tfcDefineLayerProp(
; (LayerName Purpose PropName  PropValue)
  (CNWI all function           "well")
  (CNWI all maskNumber         1)
  (CNWI all minSpacing         4.0)
  (CNWI all minWidth           5.0)
  (CNWI all minNotch           4.0)
  (CNWI all maxFilling         4.0)
```

```
(CTOX all function              "diffusion")
(CTOX all maskNumber            2)
(CTOX all minWidth              1.0)
(CTOX all minSpacing            2.0)
(CTOX all minNotch              2.0)
(CTOX all maxFilling            2.0)

(CNPI all function              "implant")
(CNPI all maskNumber            3)
(CNPI all minNotch              1.8)
(CNPI all maxFilling            1.8)

(CPPI all function              "implant")
(CPPI all maskNumber            4)
(CPPI all minNotch              1.8)
(CPPI all maxFilling            1.8)

(CPOL all function              "conduction")
(CPOL all maskNumber            5)
(CPOL all minWidth              1.0)
(CPOL all minSpacing            1.5)
(CPOL all minNotch              1.5)
(CPOL all maxFilling            1.5)

(CME1 all function              "conduction")
(CME1 all maskNumber            6)
(CME1 all minWidth              1.5)
(CME1 all minSpacing            1.5)
(CME1 all minNotch              1.5)
(CME1 all maxFilling            1.5)

(CME2 all function              "conduction")
(CME2 all maskNumber            7)
(CME2 all minWidth              1.5)
(CME2 all minSpacing            1.5)
(CME2 all minNotch              1.5)
(CME2 all maxFilling            1.5)

(CCON all function              "via")
(CCON all maskNumber            8)
(CCON all minSpacing            1.5)

(CVIA all function              "via")
(CVIA all maskNumber            9)
(CVIA all minSpacing            1.5)

(softFence drawing function     "softFence")
(softFence drawing maskNumber 10)

(hardFence drawing function     "hardFence")
(hardFence drawing maskNumber 11)

(prBoundary all function        "cellBoundary")
(prBoundary all maskNumber      12)

(CNWI  drawing lasType          "nwellLayer" )
(CPOL  drawing lasType          "polyLayer" )
(CME1  drawing lasType          "met1Layer" )
(CME2  drawing lasType          "met2Layer" )
)
```

```
tfcDefineTwoLayerProp(
; (Layer1Name   Purpose1 Layer2Name   Purpose2 Order   PropName      PropValue)
  (CNWI         all     CTOX         all      t       minEnclosure  3.0)
  (CTOX         all     CNWI         all      nil     minSpacing    3.0)
  (CPOL         all     CTOX         all      nil     minSpacing    0.5)
  (CNPI         all     CPPI         all      nil     minSpacing    0.0)
  (CVIA         all     CPOL         all      nil     minSpacing 1  .05)
  (CVIA         all     CCON         all      nil     minSpacing    1.5)
)

tfcDefineSymWire(
; (WireName     Layer   Purpose                        DefaultWidth  MinWidth MaxWidth LegalRegion)
  (poly         CPOL    drawing                        1             1        _NA_     _NA_)
  (metal1       CME1    drawing                        1.5           1.5      _NA_     _NA_)
  (metal2       CME2    drawing                        1.5           1.5      _NA_     _NA_)
  (diffN        CTOX    drawing (CNPI drawing 1.25)    1             1        _NA_     (outside CNWI drawing))
  (diffP        CTOX    drawing (CPPI drawing 1.25)    1             1        _NA_     (inside  CNWI drawing))
)

tcSetWireProp("diffN" list("lasType" "ndiffWire"))
tcSetWireProp("diffP" list("lasType" "pdiffWire"))

tfcDefineSymWireProp(
; (WireName     PropName      PropValue)
  (poly         sheetRes      33)
  (metal1       sheetRes      0.06)
  (metal2       sheetRes      0.03)
  (diffN        sheetRes      40)
  (diffP        sheetRes      80)
)
;
; Device Section
;
; Create default device classes:
;    syEnhancement, syDepletion, syContact and syPin.
tcCreateCDSDeviceClass()

tfcDefineSymEnhancementDevice(
; (XtrName SdLayer SdPurpose GateLayer GatePurpose W L SdExt GateExt LegalRegion)
  (TransP CTOX drawing (CPPI drawing 1.25) CPOL drawing 1.25 1 1.5 1 (inside CNWI drawing))
  (TransN CTOX drawing (CNPI drawing 1.25) CPOL drawing 1.25 1 1.5 1 (outside CNWI drawing))
)

tfcDefineSymContactDevice(
; (ContactName ViaLayer ViaPurpose Layer1 Purpose1 Layer2 Purpose2
;  W L EncByLayer1 EncByLayer2 LegalRegion)
  (Cpm1  CCON drawing CME1 drawing CPOL drawing 1 1 .75 .75 _NA_)
  (Cm1m2 CVIA drawing CME1 drawing CME2 drawing 1 1 .75 .75 _NA_)
  (Cdnm1 CCON drawing CTOX drawing (CNPI drawing 1.25) CME1 drawing 1 1 .75 .75 (outside CNWI drawing))
  (Cdpm1 CCON drawing CTOX drawing (CPPI drawing 1.25) CME1 drawing 1 1 .75 .75 (inside CNWI drawing))
  ; attention, l'ordre de declaration est important
  ; pour que l'autoContact fonctionne bien
  (Csub  CCON drawing CTOX drawing (CPPI drawing .5) CME1 drawing 1 1 .75 .75 (outside CNWI drawing))
  (Cwel  CCON drawing CTOX drawing (CNPI drawing .5) CME1 drawing 1 1 .75 .75 (inside CNWI drawing))
)

tfcDefineSymDeviceProp(
; (DeviceName PropName PropValue)
  (Csub function "substrateContact")
  (Cwel function "substrateContact")
)
```

```
tfcDefineSymPinDevice(
; (name maskable layer1 purpose1 w1 layer2 purpose2 w2 legalRegion)
  (m1_t  t CME1 drawing 1.5 _NA_ _NA_ _NA_ _NA_)
  (m2_t  t CME2 drawing 1.5 _NA_ _NA_ _NA_ _NA_)
  (polyt t CPOL drawing 1   _NA_ _NA_ _NA_ _NA_)
)

symRules(
; drc((Layer1Name [Purpose1 Dev1Name]) (Layer2Name [Purpose2 Dev2Name])
;    sep/enc < distance [sameNet/diffNet])
  drc(("CNWI") sep < 8.0 diffNet)
  drc(("CNWI") ("CTOX" "all" "Cwel") enc < 0.0)
  ; ajouter implants...
  drc(("CCON" "all" "Cdnm1") ("CPOL" "all" "TransN") sep < 1.0)
  drc(("CCON" "all" "Cdpm1") ("CPOL" "all" "TransP") sep < 1.0)
  drc(("CCON" "all" "Cpm1") ("CTOX" "all" "TransN") sep < 1.0)
  drc(("CCON" "all" "Cpm1") ("CTOX" "all" "TransP") sep < 1.0)
  drc(("CTOX" "all" "Csub") ("CTOX" "all") sep < 2.0 sameNet)
  drc(("CTOX" "all" "Cwel") ("CTOX" "all") sep < 2.0 sameNet)
)

tfcDefineDeviceProp(
; (viewName     deviceName     propName     propValue)
  (symbolic     TransP         lasType      "ptr")
  (symbolic     TransN         lasType      "ntr")
  (symbolic     Cdpm1          lasType      "pd_m1")
  (symbolic     Cdnm1          lasType      "nd_m1")
  (symbolic     Cpm1           lasType      "poly_m1")
  (symbolic     Cm1m2          lasType      "m1_m2")
  (symbolic     Csub           lasType      "psub_m1")
  (symbolic     Cwe            lasType      "nsub_m1")
  (symbolic     polyt          lasType      "poly_pin")
  (symbolic     m1_t           lasType      "m1_pin")
  (symbolic     m2_t           lasType      "m2_pin")
)
```