Outils de CAO et conception structurée de systèmes intégrés sur siliciu

François-René Rougeaux

THESE

présentée par

François-René ROUGEAUX

pour obtenir le titre de **DOCTEUR**

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(spécialité : *Microélectronique*)

OUTILS DE CAO ET CONCEPTION STRUCTUREE DE SYSTEMES INTEGRES SUR SILICIUM

Date de soutenance : 2 Fevrier 1987

Composition du jury :

G. Veillon	Président
F. Anceau	
B. Courtois	
P. Gentil	
AA.Jerraya	
C. Masson	
J.P. Moreau	

Thèse préparée au sein du laboratoire TIM3

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH
Vice-Présidents : B. BAUDELET
R. CARRE
H. CHERADAME
J.M. PIERRARD

Année universitaire 1985-1986

Professeurs des Universités

E.N.S.E.E.G.

BEAUFILS	Jean-claude	LOUCHET	François
BESSON	Jean	PARIAUD	Jean-Charles
BONNETAIN	Lucien	RAMEAU	Jean-Jacques
BONNIER	Etienne	SOHM	Jean-Claude
DURAND	Françis	SOUQUET	Jean-Louis
GUYOT	Pierre		

E.N.S.E.R.G.

BARIBAUD	Michel	COUMES	André
BLIMAN	Samuel	GENTIL	Pierre
BUYLE BODIN	Maurice	GUERIN	Bernard
CHENEVIER	Pierre	POUPOT	Christian
COHEN	Joseph	SERMET	Pierre
COUMES	André		

E.N.S.I.E.G.

BARRAUD	Alain	JOUBERT	Jean-Claude
BAUDELET	Bernard	JOURDAIN	Geneviève
BLOCH	Daniel	LACOUME	Jean-Louis
BRISSONNEAU	Pierre	LONGEQUEUE	Jean Pierre
BRUNET	Yves	MASSELOT	Christian
CAVAIGNAC	Jean-François	MORET	Roger
CHARTIER	Germain	PAUTHENET	René
CHERUY	Arlette	PERRET	René
DURAND	Jean-Louis	PERRET	Robert
FOULARD	Claude	POLOUJADOFF	Michel
GAUBERT	Claude	SABONNADIERE	Jean-Claude
IVANES	Marcel	SCHLENKER	Claire
JALINIER	Jean-Michel	SCHLENKER	Michel
JAUSSAUD	Pierre		

E.N.S.H.G.

BOIS	Philippe	LESPINARD	Georges
BOUVARD	Maurice	MOREAU	René
CAILLERIE	Denis	OBLED	Charles
LESIEUR	Marcel	PIAU	Jean-Michel
TROMPETTE	Philippe		

E.N.S.I.M.A.G.

FONLUPT	Jean	ROBERT	François
MAZARE	Guy	SAUCIER	Gabrielle
MOSSIERE	Jacques	VEILLON	Gérard

U.E.R.M.C.P.P.

CHERADAME	Hervé	RENAUD	Maurice
CHIAVERINA	Jean	ROBERT	André
GANDINI	Alessandro	SILVY	Jacques

Professeurs Associés

BLACKWELDER	Ronald	ENSHG
HAYASHI	Hirashi	ENSIEG
PURDY	Gary	ENSEEG

Professeurs à l'Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis
CHATELIN	Françoise

Chercheurs du C.N.R.S.

Directeurs de recherche :

CAILLET	Marcel
CARRE	René
FRUCHART	Robert
JORRAND	Philippe
LANDAU	Ioan

Maître de recherche :

ALLIBERT	Colette	GIVORD	Dominique
ALLIBERT	Michel	EUSTATHOPOULOS	Nicolas
ANSARA	Ibrahim	JOUD	Jean-Charles
ARMAND	Michel	KAMARINOS	Georges
BINDER	Gilbert	KLEITZ	Michel
BONNET	Roland	LEJEUNE	Gérard
BORNARD	Guy	MERMET	Jean
CALMET	Jacques	MUNIER	Jacques
DAVID	René	SENATEUR	Jean-Pierre
DESPORTES	Jacques	SUERY	Michel
DRIOLE	Jean	WACK	Bernard

Personnalités agréées à titre permanent à diriger des travaux de recherche
(Décision du conseil scientifique)

E.N.S.E.E.G.

BERNARD	Claude	MALMEJAC	Yves (CENG)
CAILLET	Marcel	MARTIN GARIN	Régina
CHATILLON	Catherine	NGUYEN TRUONG	Bernadette
CHATILLON	Christian	RAVAINE	Denis
COULON	Michel	SAINFORT	Paul (CENG)
DIARD	Jean-Paul	SARRAZIN	Pierre
FOSTER	Panayotis	SIMON	Jean-Paul
GALERIE	Alain	TOUZAIN	Philippe
HAMMOU	Abdelkader	URBAIN	Georges(ODEILLO)

E.N.S.E.R.G.

BOREL	Joseph	DOLMAZON	Jean-Marc
CHOVET	Alain	HERAULT	Jeanny

E.N.S.I.E.G.

BORNARD	Guy	LEJEUNE	Gérard
DESCHIZEAUX	Pierre	MAZUER	Jean
GLANGEAUD	François	PERARD	Jacques
KOFMAN	Walter	REINISCH	Raymond

E.N.S.H.G.

ALEMANY	Antoine	MICHEL	Jean Marie
BOIS	Daniel	ROWE	Alain
DARVE	Félix	VAUCLIN	Michel

E.N.S.I.M.A.G.

BERT	Didier	DELLA DORA	Jean
CALMET	Jacques	FONLUPT	Jean
COURTIN	Jacques	SIFAKIS	Joseph
COURTOIS	Bernard		

U.E.R.M.C.P.P.

CHARUEL	Robert		

C.E.N.G.

CADET	Jean	NIFENECKER	Hervé
COEURE	Philippe (LETI)	PERROUD	Paul
DELHAYE	Jean-Marc (STT)	PEUZIN	Jean Claude(LETI)
DUPUY	Michel (LETI)	TAIEB	Maurice
JOUVE	Hubert (LETI)	VINCENDON	Marc
NICOLAU	Yvan (LETI)		

Laboratoires extérieurs

C.N.E.T.

DEMOULIN	Eric	MERCKEL	Gérard
DEVINE	R.A.B.	PAULEAU	Yves
GERBER	Roland		

++++++++++++++++

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur	: M. M. MERMET
Directeur des Etudes et de la formation	: M. J. LEVASSEUR
Directeur des Recherches	: M. J. LEVY
Secrétaire Général	: Mle M. CLERGUE

Professeurs de 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Métallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique-Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint Etienne

VERGNAUD	Jean Maurice	Chimie des Matériaux & chimie industrielle

****

AVANT-PROPOS

Je tiens à exprimer ma reconnaissance à :

Monsieur Gérard Veillon, Professeur et Directeur de l'ENSIMAG, pour l'honneur qu'il me fait en présidant le jury de cette thèse.

Monsieur François Anceau, Ingénieur à Bull et ancien Directeur de l'équipe de Recherche en Architecture des Ordinateurs, pour cette passion des machines qu'il a su communiquer à ses élèves et chercheurs.

Monsieur Bernard Courtois, Directeur de l'équipe de Recherche en Architecture des Ordinateurs, pour avoir repris une équipe et ses projets en leur état.

Monsieur Pierre Gentil, Professeur ENSERG et Directeur du Centre Interuniversitaire de Microélectronique de Grenoble, pour avoir su organiser un enseignement de qualité en microélectronique à l'ENSERG.

Monsieur Ahmed Amin Jerraya, Chargé de Recherche, pour sa participation à ce travail et pour avoir accepté de lire et corriger cette thèse.

Monsieur Christian Masson, Chef de service CAO VLSI à Bull, pour tous les conseils et pour l'expérience qu'il nous a apporté en toute amitié.

Monsieur Jean-Pierre Moreau, Chef du service CAO à Thomson DCI, pour avoir accepté d'être rapporteur de cette thèse et pour le soutien apporté à ce projet.

Je tiens à remercier l'ensemble des personnes avec lesquelles j'ai été amené à travailler ou vivre au cours de ces dernières années, qu'elles soient aujourd'hui sur la rive gauche, la rive droite de l'Isère, ou en région parisienne.

Enfin, je remercie ceux qui ont eu la patience de me supporter au cours de ces dernières années, ce qui n'est pas toujours aisé.

RESUME

La représentation symbolique des masques de circuits intégrés est connue depuis environ quinze ans, mais n'est développée que depuis quelques années. Presque tous les systèmes de CAO développés récemment font usage d'une représentation symbolique des masques de circuits afin de simplifier la conception.

Dans cette thèse, un système symbolique hiérarchique de dessin des masques est décrit. Un langage de description et un environnement graphique sont présentés. L'environnement graphique est utilisé pour l'édition de cellules et de blocs, hiérarchiquement. Le langage textuel peut être considéré comme le langage cible de tous les outils de synthèse ou de compilation qui sont développés dans le cadre d'un système intégré de CAO VLSI.

Une représentation métrique des circuits est donnée, et un "Assembleur de Silicium", qui permet le dessin hiérarchique de cellules fonctionnelles en accord avec les structures d'interconnexions de base, est décrit. Par un programme Lisp le concepteur définit comment les cellules feuilles doivent être assemblées pour réaliser la fonction désirée. Ainsi des cellules fonctionnelles peuvent être générées, avec une représentation symbolique et une fonctionnalité déterminée.

MOTS CLEF

Systèmes CAO, Bases de données, Systèmes symboliques, Description procédurale, Assembleur de silicium, DRC, Editeur, Simulateur.

ABSTRACT

Symbolic Design has been around for over fifteen years, but has gained popularity only in the last few years. Most of the CAD systems recently developped make use of a symbolic representation of the different technological elements in order to make Integrated Circuits' mask layout design easier.

In this thesis, a hierarchical symbolic layout system is described. Graphical environments and a textual description are provided. Graphical environments are used for editing cells, and for editing blocks hierarchically. A textual language may be used as the target language of all synthesis and compilation tools to be developped in the framework of an integrated VLSI CAD system.

A metric representation of the layout is described and a "Silicon Assembler", which allows hierarchical design of functionnal cells according to basic interconnection structures, is presented. A Lisp program defines how basic leaf-cells must be assembled to realise the desired function. Then functionnal cells may be generated, with soft topology and soft functionnality.

KEY-WORDS

CAD systems, Data bases, VLSI, Symbolic systems, Procedural description, Silicon assembler, DRC, Editor, Simulator.

OUTILS DE CAO ET CONCEPTION STRUCTUREE DE SYSTEMES INTEGRES SUR SILICIUM

INTRODUCTION

Les progrès technologiques ayant été considérables au cours de ces dernières années, il est aujourd'hui techniquement possible de concevoir des systèmes intégrés sur silicium.

Les circuits réalisés ont une complexité qui peut être caractérisée par le nombre de transistors utilisés, à savoir plusieurs centaines de mille. Ce nombre est élevé, et ne va pas sans poser des problèmes.

Les outils de conception, face au volume d'information à traiter, connaissent une évolution vers des systèmes de plus en plus sophistiqués. Les maîtres mots sont la conception structurée, la génération algorithmique. Il est vital de pouvoir maîtriser l'ensemble des données qui peuvent être manipulées lors de la conception d'un circuit.

Après avoir présenté rapidement les divers niveaux de description d'un circuit et la méthodologie de travail souhaitée, l'ensemble des outils, qui permettent de décrire un circuit intégré en terme de masques, est abordé.

A partir de techniques de programmation relativement récentes, une approche sur le formalisme du coeur d'un outil de CAO-VLSI, et les outils de dessin de masques, la représentation hiérarchique et la construction algorithmique de circuits sont développés.

PREMIERE PARTIE

METHODOLOGIE DE CONCEPTION, ETAT DE L'ART

1 - Principes de conception d'un circuit

Le problème actuel de la conception de circuits est la gestion de l'ensemble des données générées, leur volume devenant toujours plus important avec une intégration de plus en plus poussée. Il est donc nécessaire d'adapter les concepteurs et les outils de travail à cette évolution pour maîtriser la conception des circuits intégrés de demain.

La méthodologie de dessin structuré (114) est une approche du problème. La complexité des circuits est maîtrisée par la recherche de régularité et de structuration. La complexité des systèmes est gérée par l'utilisation de techniques de dessin hiérarchiques (142). A la structuration hiérarchique des dessins s'associe la réalisation d'outils travaillant sur des structures hiérarchiques, seul moyen d'obtenir des algorithmes performants.

1.1 - Niveaux d'abstraction

Il est possible de décrire un circuit à différents niveaux. Cependant il est impossible d'appréhender tous les aspects d'un circuit. Il faut donc faire des abstractions. Les niveaux de traitement classiques sont: le niveau comportemental, le niveau fonctionnel, le niveau architectural, le niveau structurel, le niveau logique, le niveau interrupteur (MOS), le niveau électrique et le niveau topologique.

Leurs frontières, leur ordre ou même leur recouvrement ne

sont pas boujours très bien établis. Mais ce découpage permet de réduire la complexité des problèmes à manipuler.

Trois domaines sont reconnus comme importants dans la spécification d'un VLSI (32), le domaine physique, le structurel et le comportemental. Il est possible de passer d'un niveau à l'autre par une suite d'actions. Le passage d'un niveau à l'autre dans le sens descendant se traduit par l'ajout d'informations, puisque chaque niveau comporte à la fois l'information qui lui est propre, et l'information de tous les niveaux qui l'ont précédé. Chaque niveau doit donc vérifier toutes les contraintes imposées par les niveaux supérieurs en supplément de ses propres contraintes. Les contraintes les plus classsiques sont (129): les conformités fonctionnelles et comportementales du circuit (consommation, vitesse, etc...), le respect des règles de dessin liées à une technologie, la faible surface souhaitée pour des raisons de fiabilité, de performance et de fabrication, la limitation de l'ensemble des coûts de conception, validation, test d'un circuit.

1.1.1 - Description et simulation

L'utilisation d'un support commun à tous les outils utilisés facilite les interactions entre outils agissant sur les différents niveaux. De plus un circuit se génère par étapes successives plus ou moins rapides, avec des retouches plus ou moins importantes suivant ce que l'on génère, et suivant les outils de simulation utilisés. Il y a de nombreux simulateurs adaptés à chaque niveau. Les niveaux les plus connus sont: l'électrique (SPICE - 121), le switch-level (MOSSIM - 31). L'idéal serait de pouvoir simuler un circuit en utilisant le simulateur permettant d'assurer le

fonctionnement de chaque bloc rencontré, et à chaque niveau d'abstraction.

L'utilisation d'un support commun permet la localisation des actions, et le maintien d'une cohérence d'action. Aujoud'hui, de nombreux outils utilisent CIF (156) ou EDIF (61) comme support du dessin de masques. Ce type de description est à un niveau trop bas pour satisfaire toutes les contraintes rencontrées. Pour les deux autre niveaux il n'a pas de standard veritable.

1.1.2 - Représentation hiérarchique

Il existe une représentation hiérarchique possible dans les trois domaines, et il importe que les descriptions se recoupent. Une méthode est d'unifier les trois représentations en une seule hiérarchie.

Au niveau physique, les circuits sont décrits sous forme de rectangles dans la matière courante.
Au niveau structurel, le dessin est exprimé en termes de composants et de listes de connexions. Les composants peuvent être, soit des transistors, soit des instances d'autres composants (172). Les formes traditionnelles de représentation structurelle, sont les diagrammes logiques.
Au niveau comportemental, un dessin est exprimé en terme de fonctions, exemple ISPS (14) au niveau transfert de registres (RTL), LMS au niveau fonctionnel (69) ou SIMULA (23) au niveau langage de programmation.

1.1.3 - Recherche de cohérence entre descriptions

Un système intégré a pour but, le contrôle de la

correspondance entre les diverses représentations (32). Lorsque deux descriptions sont proches, il est possible de déduire l'une d'elles à partir de l'autre. Cependant, il demeure très difficile de faire du contrôle de cohérence entre niveaux différents de façon automatique. Il n'y a souvent qu'un simple recoupement, et la vérification demeure complexe.

1.2 - Conception structurée

Dans la génération des circuits VLSI, un des principaux problèmes rencontrés est le routage entre blocs fonctionnels. L'idée d'une conception descendante, structurée, hiérarchique et modulaire contribue grandement à l'améliorer.

Les trois critères que l'on associe volontiers à cette idée sont la régularité, la modularité et la localisation.

1.2.1 - La régularité

L'utilisation de la régularité permet de diminuer fortement le temps de conception. La fiabilité est augmentée par la vérification implicite ou explicite des règles de conception.

La régularité réduit la complexité du dessin et améliore l'intégration. Elle nécessite cependant une stratégie d'interconnexions. Il est possible de connecter des cellules par aboutement dans le bit-slice d'une partie opérative à condition d'avoir conçu des cellules adaptées. D'autre part, lorsque des cellules n'ont pas été conçues pour être assemblées, le coût de leur assemblage par routage est

environ 20% supérieur à celui de cellules conçues pour être assemblées par "stretch" et aboutement dans le cas de petites cellules (32). Il y a donc nécessité de s'astreindre à une discipline si on souhaite la régularité.

1.2.2 - La modularité

La réduction de la complexité par découpage se traduit, dans la conception des circuits, par la division du circuit en blocs.

1.2.2.1 - Le découpage par modules

La modularité permet le partitionnement d'un circuit, avec tous les intérêts du partitionnement. Chaque module est défini avec une spécification fonctionnelle très précise. Les outils adaptés, ou les concepteurs compétents peuvent lui être appliqués. Le temps de conception est fortement diminué et le contrôle de la complexité augmenté.

Le partitionnement et la hiérarchisation d'un circuit possèdent de gros avantages. Parmi ceux-ci, il convient de citer la possibilité de traiter un circuit à un niveau de détail ou d'abstraction choisi, ce qui est très intéressant pour un grand nombre d'outils.

1.2.2.2 - Le choix d'une méthodologie

Une des influences les plus connues dans les outils de dessin de circuits intégrés, est la méthodologie de dessin structuré de Caltech, caractérisée par un ensemble d'outils (166), expliquée de la façon suivante:

-Une méthode simple de conception de dessin de masques,

est de les dessiner à la main, puis de les digitaliser.

-La méthode "dessin à la main" et digitalisation à l'aide d'un langage symbolique représentant les masques est très pratique pour générer les fichiers dessin pour des circuits très structurés.

La fonction d'un langage symbolique de dessin des masques est, dans sa forme la plus simple, similaire à un macro-assembleur.

-Par l'utilisation de "symboles", les systèmes de layout peuvent être décrits de manière hiérarchique, conduisant à des descriptions très compactes de dessins structurés.

-La construction d'un système interactif de layout comme ICARUS est relativement simple pour celui qui a une expérience dans le domaine graphique interactif.

-La direction à suivre pour de nouveaux développements dans les outils de dessin, est la conception d'une structure de données permettant aux fonctions de dessin autre que celles liées au layout d' agir interactivement sur la même base de données.

1.2.2.3 - Le plan de masse

Pour que le découpage d'un circuit en blocs puisse se maintenir aux différents niveaux, il est généralement réalisé sur les descriptions de haut niveau.

Il faut relier les différents blocs entre eux par des connexions. La surface du circuit doit être la plus faible possible, il est donc important de faire des optimisations pour réduire la surface, tant au niveau des blocs qu'au niveau des connexions. Il faut réaliser un bon placement des blocs pour réduire les connexions, une adaptation topologique des blocs, une distribution adéquate des points

de connexion de chaque bloc. Il en découle la nécessité de faire un plan de masse du circuit s'appuyant sur une évaluation de la surface, de la forme, et de la position des connecteurs de chaque bloc (55). On doit tirer profit de la déformabilité et de la transparence des blocs. Le résultat donne des circuits très performants dont le rendement en production est très élevé.

1.2.2.4 - Utilisation des hiérarchies séparées

Il apparaît que la structuration essentielle, est la représentation d'un circuit sous forme de cellules de base, et de hiérarchies de cellules de composition. La description de circuits par arbres binaires ou algorithmes devient possible (41,92,116,169).

Une cellule de base est atomique, n'a pas de structuration hiérarchique interne. Elle peut avoir de multiples représentations, dont une géométrique (constituée des masques), une logique, une électrique, etc ...

Une cellule de composition est composée de cellules de composition, ou de cellules de base, ces dernières pouvant être instanciées à n'importe quel niveau de hiérarchie.
Les cellules de composition sont construites indépendamment de la technologie. Ce sont les règles de composition qui dépendent de la technologie. Le dessin construit est correct par construction, la vérification est facilitée, car adaptée au type de cellule traitée. De plus, les cellules de composition assurent la régularité, la modularité et la localité en empêchant la création de fonctions lors de la compositon.

1.2.3 - La localisation des problèmes

Un point important de la structuration est la localisation des données et des signaux. Ceci permet de contourner le problème des bus globaux et des routages globaux. La localisation se fait par la spécification des interfaces entre blocs comme contrainte de génération.

1.3 - Les approches de conception

1.3.1 - Les deux approches possibles

La conception est généralement abordée sous deux aspects: la démarche **"Top-Down"**, et la démarche **"Bottom-Up"**. Dans la méthodologie de conception "Bottom-up", en partant de cellules de base on assemble jusqu'au circuit final. Dans la démarche "Top-Down", des modules correspondant à des fonctions sont déclarés et le plan de masse du circuit est prédéterminé. Une surface est allouée à chaque bloc qui peut alors être généré de manière hiérarchique, ou par générateur.

1.3.2 - La conception hiérarchique descendante

La démarche de conception "Top-Down" permet de maîtriser de façon beaucoup plus efficace le plan de masse du circuit réalisé. Il est décomposé en sous plans auquels sont associés une surface, une position et des spécifications d'interconnexion et de fonctionnalité. L'opération est recommencée au niveau de hiérarchie inférieur, jusqu'au niveau du layout réel.

1.3.2.1 - L'évaluation des blocs

Le moyen le plus efficace pour avoir une telle approche du problème posé par la réalisation de circuits complexes, est l'utilisation d'une base d'évaluation de l'aire de chaque bloc à réaliser, en fonction de sa fonction, et de ses interconnexions (55,158). Il est alors possible de contrôler efficacement la taille d'un bloc, et de générer rapidement un plan de masse correct et efficace, en évitant les retours en arrière et les corrections après génération.

1.3.2.2 - Les choix topologiques

La stratégie de plan de masse et de placement est guidée en général par les interconnexions.

Le fait de considérer les blocs sous forme rectangulaire simplifie grandement le travail et permet la réalisation d'outils très performants et bien adaptés (123). L'observation des circuits complexes réalisés montre qu'il est souvent possible de les décomposer sous forme de hiérarchies de rectangles.

1.3.2.3 - La méthodologie descendante associée

La conception "Top-Down" peut se traduire par la réalisation d'un plan de masse, puis par un passage au niveau de chaque bloc, et de progresser ainsi jusqu'au niveau du dessin de cellules. La progression est telle qu'un niveau inférieur n'est pas traité tant que le niveau supérieur correspondant n'est pas réalisé. La manipulation d'un plan de masse hiérarchique est un problème complexe (158), aussi un choix de représentation sous forme de graphe polaire est

souhaitable (123).

La principale difficulté rencontrée se décompose en deux étapes:

-faire descendre les spécifications rencontrées jusqu'au niveau du layout. Les générateurs automatiques de blocs avec contraintes de surface et de position des interconnexions sont alors les bienvenus, et peuvent directement remplacer les évaluateurs lorsque le temps de génération n'est pas rédibitoire (en minutes pour des systèmes comme PAOLA (49) ou APOLLON (76)).

-lorsque une spécification ne peut être tenue, renverser la démarche de conception et remonter les contraintes rencontrées jusqu'à leur résolution par génération d'un nouveau jeu de spécifications.

1.4 - Intégration des outils de conception

Idéalement, ce que le dessinateur produit à un haut niveau d'abstraction devrait apparaître dans le "layout". Ceci peut se faire à partir d'une structure d'accueil indépendante du niveau de travail et dans laquelle tous les outils de conception, test, validation peuvent venir travailler. L'intégration d'un système autour d'une forme permet de détecter les erreurs de génération ou les incohérences entre représentations.

1.4.1 - Spécification du noyau commun

Un noyau commun est composé d'une hiérarchie de cellules feuilles ou de composition. Les cellules de composition n'ont pas de fonctionnalité propre, elles ne sont qu'un répertoire des cellules composées et d'interconnexions.

Cependant, dans leur manipulation, il ne doit pas y avoir de différence entre les deux types de cellules.

Toutes les cellules doivent être considérées comme des boites, et leurs connecteurs peuvent être positionnés au bord de ces boites. Les connexions se font par aboutement horizontal et vertical. Un routage complexe entre cellules peut toujours être considéré comme une cellule intermédiaire. Les règles de composition n'ajoutent pas de fonctionnalité. Toutes les fonctions viennent des éléments assemblés.

Les cellules de base ont divers aspects. Aussi, il est nécessaire de pouvoir vérifier la cohérence entre ces diverses représentations. Au niveau des cellules de composition, ce problème est beaucoup plus simple, l'obstacle le plus général étant la génération d'un "layout" juste. On se rend alors compte que la vérification d'un circuit devient beaucoup plus simple.

1.4.2 - Processus de génération des données

Dans l'utilisation d'un noyau commun, il convient de séparer les outils de création ou d'édition de la structure de données, des outils de vérification. Les uns ont pour soucis d'agir en parfaite cohérence et les autres de pouvoir accéder à la base en lecture. La génération des données suit généralement le processus suivant (8) dans la démarche "Top-Down", laquelle est incontestablement la mieux adaptée au problème:

-une étape d'évaluation topologique qui permet de construire le plan de masse du futur circuit (Ce plan de

masse décrit la forme des blocs, leurs interconnexions et les contraintes topologiques auxquelles ils sont soumis. Il est l'information topologique de haut niveau qui dans un système intégré de CAO est utilisé pour définir l'organisation de la base de données et la gestion de la conception du circuit complet),

-une étape de conception logique et électrique de chaque bloc (La taille des transistors est calculée à partir de l'évaluation des longueurs de connexion dans le plan de masse),

-une étape de conception indépendante de chaque bloc (Le plan de masse, les schémas électriques et logiques fournissent les spécifications de conception pour chaque bloc. Ils forment la structure de gestion de données pour la conception, où toutes les modifications sont reportées).

-enfin, il y a assemblage final des blocs en accord avec les indications du plan de masse.

2 - Différents niveaux des outils de conception

Un système de conception de circuits intégrés de grande complexité peut se décomposer en deux familles d'outils:

-les outils dits de "base", qui sont capables de permettre manuellement le dessin, la simulation ou le test d'un circuit intégré à partir d'une description précise.

-les outils dits "intelligents", qui sont capables de

prendre des décisions ou appliquer des stratégies choisies et caractérisées par des architectures cibles, des méthodologies de conception.

2.1 - Outils de base

Au niveau de ces outils, se trouvent généralement:

- un éditeur graphique permettant la gestion des structures générées.
- une structure de données correspondant à la représentation des masques des circuits. A cette représentation peuvent être associées des informations sur l'aspect électrique du circuit généré.
-des outils de dessin de masque, à partir de l'éditeur graphique et de la structure de données. Cet éditeur peut être incrémental, manipuler des représentations symboliques, faire un certain nombre de vérifications sur la justesse du dessin généré, ou sur sa cohérence vis à vis d'autres représentations.
-des outils de simulation, la plupart travaillant soit au niveau transistor, soit au niveau logique. Il est nécessaire de concevoir une structure de donnée adaptée, ou d'envisager l'existance d'extracteurs qui à partir du dessin des masques génèrent un graphe électrique.
-des outils de test, basés sur le principe de l'analyseur logique, du vecteur de test, ou de l'observation par microscope électronique.
-des outils de comparaison qui permettent de verifier qu'une représentation à un niveau donné est identique à une représentation sur un autre niveau.

2.2 - Outils ayant un niveau de décision

Les outils ayant un niveau de décision font appel, en général, aux capacités d'"intelligence" de l'ordinateur par un moyen ou un autre. Les méthodes les plus répandues sont généralement la réalisation de moteurs d'inférence dans les techniques de l'intelligence artificielle, la réalisation d'algorithmes complexes, ou des procédures spécialisées dans l'interprétation des données avec un objectif parfaitement déterminé.

Ce type d'outils se caractérise par des extensions d'outils de base. Les structures de données manipulées et qui caractérisent ces outils sont souvent un prolongement plus puissant des structures de données des outils de base.

2.2.1 - L'évaluation des blocs

L'approche de conception associée étant de caractère "Top-Down", il apparaît souvent la nécessité de faire des prévisions. Pour cela on utilise des évaluateurs. Un évaluateur est ce qui permet de connaître à l'avance, avec une erreur minimale, la surface des blocs manipulés, l'emplacement des connexions, les performances et la consommation des différents blocs qui composent un circuit, sans se préoccuper de la conception détaillée. Un évaluateur est réalisé à partir de données de caractère statistique, technologique, méthodologique.

2.2.2 - Les compilateurs de silicium

Les outils "intelligents" de plus haut niveau sont ceux qui permettent la conception de circuits complexes à partir de spécifications de haut niveau. Ces outils, appelés

compilateurs de silicium, sont classés en plusieurs familles, les compilateurs d'architecture comme GENESIL, les compilateurs d'algorithmes comme MACPITTS (152), SYCO (78).

Du point de vue architecture, les circuits générés sont, soit des circuits pré-diffusés, soit des circuits analogiques, soit des circuits dits "full-custom".

Du point de vue fonctionnel, ils servent au filtrage et au traitement du signal, ou ont un comportement algorithmique. Dans ce dernier cas, l'architecture cible se décompose en une partie opérative (chemin de données) et une partie contrôle. On génère ainsi des microprocesseurs, un certain nombre de co-processeurs, les contrôleurs de périphériques, de communications, les séquenceurs, ...

2.3 - Validation et test des circuits

Dans les outils d'aide à la conception, les outils de validation et de test prennent une part de plus en plus importante, qu'il s'agisse du test de validation, du test de fabrication, ou du test en ligne. Le lien entre outils de CAO et de test (analyseurs, microscope) est réalisé (69-1).

2.3.1 - Test de validation

Une méthode utilisée est la possibilité de comparer des points d'un circuit en fonctionnement à l'aide d'un microscope, avec les mêmes points tels qu'ils sont décrits et simulés (69-1). Des erreurs de fonctionnement liées à la conception peuvent ainsi être détectées.

2.3.2 - Test en ligne

Son but est d'assurer au niveau matériel la détection immédiate de l'apparition d'une erreur en cours de

fonctionnement.

2.3.3 - Test hors ligne

Il sert en grande partie à la maintenance des circuits. A partir de cellules spécifiées, l'ensemble des séquences de test qui permettent de voir si, une fois en fonctionnement, le circuit ne connaît pas de problèmes de fonctionnement, sont déterminées. En cas de panne, la localisation de la panne et ses conséquences sur le fonctionnement du circuit sont déterminées. Il existe aussi le test de fabrication qui a pour rôle le contrôle du processus de fabrication, et la détermination des paramètres électriques réels.

2.4 - Situation actuelle de la CAO-VLSI

Tous ces outils nécessitent que la représentation et la structure associées aux outils de base soit suffisamment bien établies, souples et puissantes. Les systèmes existants sont extrêmement nombreux, (plusieurs centaines) (cf. annexe 1), ont des degrés de performance plus ou moins importants et ne sont pas toujours adaptés aux problèmes rencontrés. Les caractéristiques d'un certain nombre de systèmes vont être étudiées. Il demeure cependant certain que peu de systèmes voient le jour de façon industrielle et définitive. Ce marché est extrêmement réduit, et les critères de choix d'un système, par les industriels, sont avant tout déterminés par la fiabilité et la stratégie. De nombreux concepts et systèmes existants, extrêmement intéressants, ne trouvent pas ou peu d'écho auprès des fabriquants ou concepteurs de circuits.

3 - Outils de génération des circuits intégrés

Différentes façons de générer les circuits existent, liées à la méthode de conception choisie.

La méthode peut être manuelle (dessin à la main) ou automatisée à l'aide de langages de programmation de description textuelle.

Le niveau d'abstraction dans la représentation d'un circuit peut varier énormément suivant que l'on manipule les masques ou que l'on travaille à un niveau d'abstraction élevé (représentations symboliques).

Les outils existants sont classés suivant leurs caractéristiques.

3.1 - L'approche symbolique

La représentation d'un circuit se traduit, en général, par des cellules ayant un nom, et une liste d'attributs, ces attributs pouvant être des points de connexion sur le contour d'une "Bbox" représentant la cellule, un schéma électrique et le dessin des masques.

L'approche qui incontestablement a changé beaucoup de choses au domaine de la conception, est celle développée dans les années 70, et présentée par J.D.Williams (187).

Cette approche consiste à représenter les masques de circuits sous forme de symboles. Il est alors possible d'avoir sur une même description l'aspect électrique et l'aspect topologique du circuit.
Les systèmes développés autour de cette approche se sont énormément développés et diverses méthodologies ont vu le jour. Ainsi, Silicon Design Labs avec le langage L (34) crée un ensemble intégré où l'aspect électrique et topologique

sont unifiés dans une même base de données. A l'opposé, Phoenix Data Systems Inc. utilise des bases de données séparées pour les diverses représentations et intègre l'ensemble de ses outils par comparaisons entre les diverses représentations (159).

3.2 - Spécification des problèmes à traiter

En règle générale, les stations de travail utilisent pour outils de génération un éditeur graphique au niveau du masque, un éditeur au niveau symbolique, et des compacteurs. A ceci il convient d'ajouter tous les outils de conception basés sur des représentations textuelles et algorithmiques. Ainsi, le cahier des charges d'un outil comme ELECTIC (143), présente comme arguments à la conception:

- Pour une plus grande flexibilité du dessin, une spécification top-down des circuits doit être disponible, ainsi, de larges parties d'un circuit peuvent être manipulées avant que leur contenu ne soit défini.

-Un des problèmes des outils de conception est leur incapacité à admettre les nouveaux outils d'aide et d'analyse qui apparaissent régulièrement. La génération d'un "layout" s'accompagne d'un nombre important de pré-processeurs et post-processeurs. La maîtrise de l'ensemble de ces processeurs devient un veritable sport en raison des différentes syntaxes ou des différentes interfaces. De plus, bien souvent, s'il est possible de passer d'un processeur à l'autre, l'absence d'intégration rend les retours en arrière extrêmement lourds et critiques (simulation, génération, extraction, resimulation, ...).

- La génération d'un "layout" peut être vue sous deux aspects différents. Tout d'abord, l'interface de génération peut être soit graphique, soit textuel. En suite, on peut vouloir privilégier l'aspect connectique, ou l'aspect géométrique.
La description textuelle insite à la génération algorithmique avec les problèmes de cohérence du dessin que cela entraîne, alors que l'environnement graphique pousse à la génération incrémentale, avec suivi des erreurs de conception. La description textuelle peut devenir très puissante avec la notion de langage immergé, le paramétrage et la documentation. De plus le problème graphique est alors un simple problème d'affichage. L'édition graphique est simple d'emploi et permet des modifications extrêmement rapides.

- Un circuit peut être représenté soit sous forme "sticks" de connectivité, ou sous forme de placement géométrique. Dans les systèmes "sticks", l'information électrique est présente, il n'y a pas à l'extraire pour les outils d'analyse électrique. L'inconvenient principal de ces systèmes, est que le concepteur n'a pas la maîtrise de la génération fine du layout. Les systèmes géométriques sont beaucoup plus simples, mais toutes les informations d'ordre électrique ou fonctionnel doivent être calculées.

- Un environnement de dessin doit pouvoir supporter toute nouvelle approche de dessin, toute modification technologique, et il doit fournir un environnement uniforme de dessin.

3.3 - Description symbolique des masques

Dans une description symbolique, tous les éléments tels que les transistors, les contacts, sont représentés par des symboles. Un circuit intégré est alors représenté sous forme d'un graphe.

Les composants électriques manipulés sont représentés par les noeuds d'un graphe. Les liaisons électriques qui les relient sont les arêtes du graphe. Chaque noeud ou chaque arête d'un graphe possède ses propres informations électriques. Dans la hiérarchie d'un dessin, un graphe peut s'exprimer à un niveau supérieur sous forme d'un noeud complexe. Les circuits sont décrits de manière hiérarchique et mis dans des bibliothèques. Dans une démarche "Top-Down", la propagation des modifications sur une cellule de base se fait automatiquement, afin d'assurer la cohérence des connexions. L'algorithme utilise pour cela les typages faits sur les fils de connexion.

Au niveau des composants de base, il y a des noeuds et des arêtes dits primitifs, qui sont décrits pour une technologie (MOS, bipolaire, circuit imprimé). En MOS, les noeuds utilisés sont le transistor, le contact, et les fils de métal, poly ... En bipolaire, les noeuds sont les différents éléments d'un transistor, la base, l'émetteur, le collecteur, et les connexions. Pour les circuits imprimés les noeuds sont les librairies de circuits et les composants logiques, les connexions. A chaque type de noeud d'une technologie donnée est associé un certain nombre de propriétés, comme leur description fonctionnelle, leur connectivité, leur taille par défaut, et leurs aspects géométriques.

En général, les systèmes de représentation symbolique sont de trois types (122):

- les grilles métriques comme les premiers systèmes SLIC (67) ou FATFREDDY (15),
- les grilles relatives comme il y en a dans les systèmes SLIP (60), STICKS (186),
- les grilles virtuelles, principalement développées par N.WESTE, que l'on trouve dans des outils comme MULGA (181) ou MGX (127).

3.3.1 - Les systèmes métriques

Une cellule est représentée telle qu'elle sera à partir de masques abstraits ("logs" dans MAGIC (125,126)), c'est à dire de "layout" symbolique métrique où les objets sont représentés en position et dimension réelle, cas de STYX (79). Ceci veut dire que le concepteur travaille à un niveau de raffinement très proche du masque. Il peut arriver à une grande densité d'intégration, mais perd beaucoup de temps. Les premiers systèmes conçus utilisaient des grilles fixes dont le pas était donné par la technologie. Ils présentaient un énorme problème, car le concepteur ne pouvait rien dessiner en dehors de cette grille aux mailles trop laches. Aujourd'hui, les progrès effectués au niveau du matériel informatique permettent de pallier à cet inconvénient dont la principale raison était l'impossibilité pour les systèmes de traiter de nombreuses données aux dimensions excessives. Il est établi qu'il est possible de travailler sans aucun problème en prenant pour pas de grille le centième de micron, avec les technologies actuelles. Cependant, si l'utilisation du métrique est intéressante dans la mesure où nous ne faisons pas appel à l'intelligence de l'ordinateur, où nous n'utilisons pas de post-processeur

au dessin, et où des vérificateurs incrémentaux peuvent exister, les contraintes imposées au dessinateur et le temps perdu à la conception doivent être mesurés en relation avec la nécessité d'une telle intégration.

3.3.2 - Les systèmes à grille relative

Ces systèmes sont le contre-pied du précédent. Ici, le but essentiel est de concevoir le plus rapidement possible en soulageant le concepteur des taches de placement fin des objets. Il se contente de donner les positions relatives et le type d'interconnexions qu'il souhaite réaliser.
Une fois une cellule ainsi décrite, un logiciel de compactage (47) comme dans CABBAGE (73), PYTHON (12), à partir d'une analyse du chemin critique, ou AMOEBA (104), REST (119,120), à partir de la résolution du graphe des contraintes (105), génère un format de sortie correspondant au dessin des masques (CIF (156), EDIF (61), forme stick standard (165)). Les stations de travail symbolique les plus utilisées travaillent avec ce principe, que ce soit VTI (167,168), CALMA (38), PSI (64), HILL (99).

3.3.3 - Les systèmes à grille virtuelle

Ce type de représentation consiste à rendre un système aussi fin que le premier quant à la densité de conception, et aussi pratique et souple quant aux possibilités de déformation d'une cellule que le second. Ici, l'espace entre deux mailles d'une grille n'est pas fixé en absolu. Il peut varier suivant le contexte de façon non uniforme. En fonction des éléments placés sur une ligne, un pas de garde est calculé, en fonction de la technologie. L'espace final entre les grilles n'est calculé que lors de la réalisation

du masque du circuit, en fonction du contexte. Pour cela des algorithmes de compaction sont définis (5).

3.3.4 - Exemples de représentations des masques

En étudiant l'ensemble des outils développés à CALTECH, un aperçu de toutes les familles d'outils est donné: LAP (103), REST (119,120), PAUL et SAM (169), SPARCS (35).

LAP (103): Il permet de dessiner les cellules de base. Il est un outil de dessin géométrique de bas niveau. Toutes les primitives sont écrites sous forme de rectangles dans la matière courante. On définit ainsi des procédures qu'il suffit ensuite d'appeller. LAP est immergé dans le langage SIMULA (23), sa sortie principale est CIF (156) et sert à la fabrication et aux vérificateurs de règles de dessin.

REST (119): Il permet également de dessiner des cellules feuilles, mais il est basé sur les principes du layout symbolique sous forme stick (186). REST utilise un interface graphique de haut niveau. Il comporte le descriptif topologique et électrique d'un circuit, mais ne comporte pas l'aspect comportemental. Cet outil a pour sortie le format stick standard (165) qui sert aux outils de composition.

PAUL: Il est très proche de LAP, immergé en SIMULA, mais a pour sortie le format stick.

SAM (169): Il est le dernier outil de dessin de cellules feuilles. Il est écrit en SMALLTALK (68) et combine l'entrée graphique avec un langage de layout. Les cellules peuvent être paramétrisées, ou définies de manière algorithmique.

SPARCS (35): il s'agit du dernier outil développé. Plus qu'un outil, il s'agit d'une base intégrée de CAO, dans laquelle le principe choisi est la représentation objet "système OCT" (35) où l'interface graphique est géré par un système autonome, "système VEM" (35). Le "layout" est représenté sous forme de symboles et un graphe des contraintes est construit. La réalisation du "layout" est faite lorsque le graphe des contraintes est résolu. SPARCS, comme les outils développés au MIT fait largement appel aux techniques de représentation objet (chapitre suivant) et aux méthodes développées autour des langages de haut niveau.

3.4 - Méthodes de génération du layout

Hormis la méthode purement manuelle et graphique que l'on retrouve dans pratiquement tous les systèmes avec les développements qui en résultent au niveau des éditeurs de cellules, de hiérarchies ou de plan de masse, l'autre méthode de conception qui se développe de manière irréversible, est l'utilisation de l'informatique dans l'art de compiler, analyser ou évaluer un texte. Le problème de la conception devient un problème d'édition de texte.

3.4.1 - Approches textuelles de génération

Trois approches textuelles peuvent avoir lieu: la génération de descriptions statiques, la génération de descriptions paramétrées à caractère algorithmique ou l'immersion du langage dans un langage de programmation classique. Les avantages des trois méthodes sont exposés dans le chapitre suivant.

L'utilisation d'une représentation textuelle pour décrire

des cellules sous les trois formes se trouve dans pratiquement tous les systèmes:
STICKS (187), CIF (156), EDIF (61), LUCIE (126) ... pour des générations statiques,
L (34), HIDELA, SILCO (130), ... pour des descriptions paramétrées,
ALI (101), DPL (16), ICPL (93), ... pour des descriptions immergées.

L'utilisation d'un langage n'est pas forcément intéressante lorsqu'il s'agit de décrire des cellules. En revanche, elle trouve tout son intérêt lorsqu'il s'agit de construire des cellules par composition, comme nous l'avons vu précédemment. Ce qui est le cas dans toutes les démarches où la construction hiérarchisée, structurée et régulière est envisagée.

Les outils, traduits par des langages de description, sont très nombreux. Leur but est toujours de construire des cellules par composition, en déclarant des assemblages de cellules, des interconnexions ou des placements relatifs. Certains se situent encore au niveau topologique, alors que d'autres se situent à un niveau d'abstraction où les algorithmes d'assemblage sont entièrement pilotés par des considérations d'ordre électrique ou fonctionnel.

3.4.2 - Exemples d'approches textuelles

Comme langages de description il est possible de citer à titre d'exemple:

3.4.2.1 - Les outils développés à CALTECH

BRISTLE BLOCKS (82): Il est un outil d'assemblage de cellules élémentaires, spécialisé dans la génération de parties opératives. Dessiner une cellule consiste à écrire un programme qui permet son assemblage à l'exécution. L'assemblage se fait par aboutement de cellules qui sont ajustées par "stretch". L'entrée est donc une description de haut niveau et un ensemble de cellules paramétrées. Le résultat est loin d'être optimal. Bristle Blocks ne présente ni description comportementale, ni description fonctionnelle.

SLAP/EARL (89): Ils sont des outils d'assemblage de cellule. Il utilisent des cellules avec contraintes sur les connecteurs, et la technique du "stretch". Un assemblage consiste en la résolution du graphe de contraintes généré. SLAP est immergé en SIMULA et EARL possède son propre langage qui est interprété.

SPAM: C'est l'outil de plus haut niveau. Un circuit décrit de manière hiérarchique peut être simulé à tous les niveaux. L'utilisateur choisit quel est le plus bas niveau de simulation qu'il souhaite avoir, et il peut simuler le comportement de cellules composées. La description structurelle se fait par le biais des connecteurs des cellules qui sont typés, ce qui permet de vérifier la validité des assemblages effectués. SPAM est conçu pour permettre le dessin dans une démarche TOP-DOWN.

3.4.2.2 - Outils de génération procédurale

LUBRICK (145): Il est un outil d'assemblage de cellules feuilles métriques, sans possibilité de "stretch". Les cellules doivent être conçues pour s'assembler. Les connexions sont caractérisées par des connecteurs, auxquels sont associés des directions de connexion. Les connecteurs peuvent être typés par un numéro. Le langage LUBRICK se traduit par un ensemble de fonctions PASCAL.

ALI (101): Il est un langage de conception de cellules, par écriture de procédures, dans lesquelles les tailles et positions des éléments manipulés doivent être spécifiées. ALI manipule des collections de rectangles et des relations entre ces rectangles. Il est possible d'avoir sur une description les informations de connectivité. Il n'a pas besoin d'extracteur. ALI est immergé en PASCAL. Il peut donc utiliser la possibilité de structuration hiérarchique pour représenter de grands circuits. Les principaux problèmes sont, l'absence totale d'un descriptif comportemental, le manque de primitives de dessin et de connectique qui rendent les programmes de génération complexes, et enfin les problèmes liés à l'utilisation de PASCAL (compilation non séparée, absence de types génériques et de tableaux dynamiques, ...). A ALI, est associé un langage d'assemblage de cellules ALLENDE (117,118) dans lequel les cellules sont aboutées, celles-ci pouvant avoir des contraintes quant à la position de leurs connecteurs. Dans ce cas, les cellules peuvent être ajustées par "stretch" et résolution du graphe des contraintes qui résulte des assemblages effectués. ALLENDE, comme ALI peut être appelé en PASCAL ou en C.

SCALE (33): Il est beaucoup plus flexible qu'ALI. Il se caractérise par une collection de langages spécialisés qui permettent divers types de génération automatique. Les objets principaux traités dans SCALE sont le contact, le transistor, le fil, et la cellule. Il y a séparation entre le dessin de cellules de base et cellules composées. La sortie de tous les outils de génération existant dans SCALE est une forme intermédiaire (IDL). C'est le point d'entrée des outils de génération de masques, des DRC et simulateurs. En IDL, il est possible de générer des hiérarchies de cellules "stretchables" et le mode de représentation est celui des coordonnées Buchanan (32).
Trois types de cellules existent. Les cellules de base, les cellules de composition et les cellules spéciales. Les cellules de base sont constituées des éléments de base, les cellules de composition sont réalisées par aboutement, et les cellules spéciales correspondent à des "layouts" décrits au niveau des masques. Scale ne fournit pas de description fonctionnelle, ni de description comportementale.

MacPitts (152): Cet outil est un "compilateur de silicium", écrit en lisp, pour dessiner des chemins de données séquencés par microprogrammes. Il part d'une description de haut niveau, basée sur la notion de transfert de registre. La génération d'un "layout" est faite en deux étapes: la génération d'un code intermédiaire indépendant de la technologie, une génération qui est dépendante de la technologie, en CIF. Les circuits sont décrits de façon algorithmique. Le "layout" découle d'un algorithme. Le principal inconvénient est qu'il n'admet qu'un seul type d'architecture cible, et génère des circuits de faible densité (surface occupée environ 10 fois celle d'un circuit fait à la main). Il comporte comme outils de validation un simulateur

fonctionnel.

A l'image de MacPitts, de nombreux compilateurs d'un langage de haut niveau existent et sont spécialisés dans une application bien précise. Leurs principaux problèmes sont souvent leur extrême spécialisation et la faible densité des circuits générés. Ils ne peuvent donc constituer un système CAO complet.

DPL/Daedalus design environment (17), **ICPL** (93), **NS** (45):

Fruit du Laboratoire d'Intelligence Artificielle du MIT, DPL est une nouvelle façon de voir la conception de circuits intégrés. Premier langage de description des circuits par procédures, DPL a connu un certain nombre de descendants dans l'industrie dont ICPL et NS. Il est avant tout un environnement interactif de génération de circuits. Il est un langage de description des circuits, alors que DAEDALUS est un environnement graphique de génération. Ils sont organisés autour d'une base de données hiérarchique et orientée objet. Le tout étant immergé en lisp, dans le cadre d'un langage orienté objet.

En DPL, la description d'une cellule se fait par l'écriture d'une procédure lisp qui, lorsqu'elle est évaluée, génère la structure de données correspondante. Les caractéristiques principales d'une procédure DPL sont: un nom de typage et la déclaration de paramètres pouvant avoir des valeurs par défaut, la déclaration de contraintes sur les paramètres utilisés, le corps proprement dit de la procédure, constitué essentiellement de l'appel suivant les paramètres à des procédures de base ou déja déclarées, enfin de contraintes sur le corps de la procédure. Par Daedalus, il est possible de visualiser le résultat d'une description, ou de le générer. Toute action de Daedalus peut être traduite en une

procédure lisp. Avec DPL/Daedalus la génération incrémentale et procédurale unie est abordée. L'ensemble DPL/daedalus repose sur une base de donnée orientée objet sur machine lisp (16).

SKILL (97,190) et L (34):

Langages de génération procédurale de circuits, ils sont immergés dans un environnement de programmation, IL pour SKILL et UNIX pour L (il possède son propre langage, très proche de C). Ils sont des outils très puissants pour définir des modules générateurs et la paramétrisation de ces modules. Le langage L est conçu autour d'une base de données comprenant les autres outils de CAO d'un processus de dessin, qu'ils soient fonctionnels ou structurels.

Palladio (30):

Cet outil présente la catégorie des outils sur la manipulation du layout où il y a le plus grand niveau d'abstraction, et ouvre la voie de la conception à l'aide de concepts venant de l'intelligence artificielle. Il propose une méthode de dessin à partir d'une base de connaissances, et des systèmes experts comme moyen de dessin. Il est possible de décrire des modèles de structure de circuits ou des comportements. Il est possible de fournir et ensuite de raffiner les spécifications qui vont permettre de générer des cellules de composition. Palladio possède un environnement graphique d'édition et un langage de description comportementale. La méthode de dessin associée est celle de la génération d'un circuit par raffinement successif des spécifications associés à des simulations comportementales. Cet outil apparaît plus, comme un moyen de choisir une architecture ou une représentation hiérarchique d'un circuit, que comme un véritable outil de CAO.

3.4.2.3 - Principales différences entre les langages

Différents outils ayant été étudiés, il convient de noter la différence existant entre des langages à caractère algorithmique comme ALLENDE (117) ou LUBRICK (145), ou l' on manipule, sous forme de programmes, des actions sur des cellules (connexion, assemblage), et des langages par procédures comme DPL (16) ou ICPL (93), où les objets sont directement manipulés. Il n'y a plus de différence entre la structure de donnée et le programme algorithmique de construction.

Enfin, un autre point à souligner est le fait que l'écriture de langages algorithmiques non immergés dans un langage de programmation, en général PASCAL, LISP, ou C (voire Algol dans le cas de SAGA (161)), limite considérablement les possibilités algorithmiques, comme on le voit dans des systèmes comme (HIDELA (130), HSL-FX (73)). A moins que le langage ait atteint un stade de développement comparable à celui d'un langage classique (cas du langage L de SDL qui est sous UNIX, très proche de C).

La liste des systèmes et des langages est très longue, et les diverses représentations, mécanismes d'assemblage ou algorithmes de génération ne manquent pas. Une liste non exhaustive est donnée en annexe, tout en remarquant que la conception de circuits semble aujourd'hui se diriger vers des systèmes où il est possible de développer rapidement des générateurs de blocs spécifiques à une application. Ces générateurs prennent sur eux de fournir tous les outils de simulation, validation et test associés. A ces outils semble s'ajouter l'existence d'une base de données, orientée objet, fonctionnant à partir de techniques dites "intelligentes",

et permettant la vérification de la cohérence d'une description, quels que soient les outils ayant travaillé sur la description d'un circuit.

DEUXIEME PARTIE

STATION DE CONCEPTION CAO-VLSI

1 - Problèmes liés aux outils de CAO-VLSI

Les caractéristiques d'un environnement de conception de circuits doivent être de pouvoir:

-décrire les circuits,
-les valider par simulation,
-les tester après réalisation.

1.1 - Problèmes liés aux actions de conception

Il apparaît que les outils des trois classes interfèrent à tous les niveaux. En partant de la représentation fonctionnelle de haut niveau du circuit, et en utilisant les outils de description, on enrichit la structure de données jusqu'à obtenir un dessin des masques. A chaque niveau de description correspond un type de validation ou de test. Le problème rencontré est de faire cohabiter tous les outils utilisés sachant que:

-ils interagissent entre eux. Une fonctionnalité X associée à des paramètres temporels et électriques se traduit par la génération d'un masque. La simulation correspondant à ce masque entraîne une modification, soit des paramètres, soit du masque lui-même. Il faut donc déterminer les interactions entre outils, leurs corrélations, et en déduire un système de conservation de la cohérence entre différents niveaux de description.

-le volume de données traité, ou généré, lors de la conception de circuits est gigantesque. Au niveau des masques, un circuit de 100.000 transistors représente un volume de environ un million de rectangles, soit plusieurs mégaoctets de données. Les outils de simulation utilisent un temps CPU qui peut être considérable, et le volume de données, correspondant au flux de sortie de ces outils est souvent énorme et difficile à traiter, même s'il peut être directement stocké sur disque. Enfin, le test, avec la génération de vecteurs de test s'avère être encore plus pénalisant, certaines procédures de test se mesurant en mois, voire en années.

1.2 - Intégration des outils de conception

Finalement, il apparaît que la conception se caractérise par un important volume de code généré par des outils (appelés processeurs), pouvant interagir. Or, pour qu'un processeur soit puissant, il convient qu'il soit indépendant. Des processeurs conçus pour travailler ensemble dans une base intégrée, en raison des contraintes qui leur sont infligées; ont tendance à perdre de leur puissance. Un type de données adapté à un processeur, ne l'est pas pour les processeurs voisins dans la chaine de conception.

Cependant, chaque outil développé peut toujours se caractériser par des entrées (INPUT), une action, et un résultat (OUTPUT). De ce fait, il convient de déterminer les méthodes de passage d'un outil ou processeur à l'autre, les méthodes de maintien de la cohérence entre les diverses représentations, en partant d'un noyau commun aux différents processeurs.

1.2.1 - Propriétés d'un processeur

Pour chaque processeur il convient de déterminer:

-en entrée:

-les données utilisées (contrôle sur la base)

-la cohérence des données (homogénéité)

-la capacité du processeur à les traiter,

-en sortie:

-le bilan d'action du processeur (flags STATUS)

-la validation des données obtenues

-la mise à jour de la cohérence des données éditées (possible retour au processus père, et maîtrise de la cohérence dans une pile de processus).

1.2.2 - Choix d'un mode d'intégration

La question posée est de concevoir un noyau qui permette l'intégration de l'ensemble des outils nécessaires à la conception d'un circuit, sans pour autant pénaliser l'un d'entre eux en lui appliquant des contraintes inadaptées.

Les approches basées sur une informatique traditionnelle semblent limitées. La raison essentielle est l'incapacité à structurer et formaliser l'ensemble des problèmes rencontrés (108). Depuis la fin des années 70 et le début des années 80, une nouvelle approche du problème de la conception de circuits apparaît, basée sur des concepts développés autour du langage lisp (106) et de ses descendants SIMULA (23), SMALLTALK (68).

L'intérêt que comporte cette nouvelle approche est discuté

rapidement, ainsi que les raisons pour lesquelles elle semble répondre au problème posé par la conception des circuits intégrés.

2 - Approche objet de la programmation

2.1 - Introduction

2.1.1 - Contrainte

La nécessité rencontrée avec un environnement de CAO-VLSI est de concevoir un puissant système d'information, dans lequel l'utilisateur peut sauvegarder, accéder et manipuler l'information, de telle sorte que le système croisse au rythme de ses idées.

2.1.2 - Axes de développement

Cette contrainte est traduite par la nécessité de maintenir deux principes qui sont les aires de recherche: concevoir un environnement ou langage de description (langage de programmation) et concevoir un environnement ou langage utilisateur.

2.1.3 - Méthodologie

La méthodologie la plus courante se traduit par la création d'un système englobant la compréhension des besoins logiciel, implanter un certain nombre d'applications et sur les résultats obtenus, reformuler la compréhension des besoins et revoir le langage de programmation, l'interface

utilisateur. Cette méthodologie se retrouve dans des systèmes comme SMALLTALK (68) et ses différentes versions ou dans des environnements d'écriture de systèmes experts comme LRO (136,137,138). Il est donc possible de proposer une solution au problème posé. Mais il ne peut s'agir de "la solution", l'approfondissement de tous les outils utilisés n'ayant pas été fait, en particulier en ce qui concerne les représentations de haut niveau et le descriptif comportemental. A ce sujet, une approche allant dans le même sens, suite à la conception du langage IRENE (28), a été proposée par Souheil Marine (108). Une maquette, résultat d'un premier travail au niveau des outils de représentation des masques de circuit, et des outils de simulation électrique, de vérification des règles de dessin va être exposée.

2.1.4 - Outils nécessaires à la conception

Comme cela est dit dans le système de conception ELECTRIC (143), il est supposé qu'un certain nombre d'outils est nécessaire à la conception de circuits, parmi lesquels:

* un générateur de "layout" symbolique, dessin de circuits imprimés
* un vérificateur de règles de dessin
* les simulateurs électrique fin, switch-level, logique et comportemental.
* un format d'entrée, de sortie (interface outils)
* des routeurs et plan de masse avec optimisation.
* des compacteurs.
* un générateur de vecteurs de test.
* des générateurs de structures régulières.
* un générateur de gate array pour applications

spécifiques.

* une interface conditionnement et fabrication.
* un analyseur statique de puissance consommée, vérification des tailles de transistors, de détection de court-circuit.
* un comparateur de description fonctionnelle de haut niveau et du circuit réalisé.
* une vérification des délais pour analyse des délais statiques.
* une interface langage permettant l'utilisation de bibliothèques de cellules existant.
* un environnement tuteur d'initiation au système et à ses outils.

2.2 - Les langages orientés objets et les langages acteurs

La prise de conscience du fait qu'il est parfois préférable de disposer d'une base de connaissances riche et bien structurée, plutôt que de résolutions complexes a entraîné une modification du mode de résolution de certains problèmes.

2.2.1 - Structuration des connaissances

Minsky (115) en décrivant la structuration des connaissances sous forme de "frames" ou schémas en a apporté les premiers éléments. Les structures de données sont conçues sous forme d'objets et de situations. A partir d'un objet et d'une situation il en découle une action. Cependant, deux tendances existent quand à l'utilisation de cette représentation. Soit on est amené à privilégier les relations qui existent entre les concepts, soit on est amené à privilégier les propriétés des concepts. Il y a donc, soit un formalisme rationnel, soit un formalisme objet (63).

2.2.2 - Vocabulaire et définitions

Dans les deux approches, les termes utilisés sont: l'"**objet**", la "**classe**", l'"**instance**", le "**message**" et la "**méthode**".

-Un objet est la somme d'un espace mémoire personnel et d'un ensemble d'opérations.

-Un message est une requête à un objet pour qu'il réalise une opération. Un message dit quelle opération il désire, c'est l'objet receveur du message qui décide comment la réaliser.

-L'ensemble des messages auquel répond un objet est l'interface entre l'objet et le système. Le seul moyen d'interagir avec un objet est à travers cette interface. Son espace mémoire personnel ne peut être manipulé que par ses propres opérations. L'envoi d'un message est le seul moyen d'invoquer les opérations sur un objet.

-L'implantation d'un objet ne peut dépendre des détails internes d'autres objets, seulement des messages auquels ils répondent. Les messages assurent la modularité du système, car ils spécifient le type d'opérations désirées, mais non comment elles sont réalisées.

"Un objet est une entité qui dispose d'attributs. Définir un ensemble de propriétés sur un objet revient à définir les attributs correspondants".

2.2.3 - Modes de représentation des données

A partir de ces données, il est possible de concevoir la programmation et les modèles de représentation des connaissances de différentes manières: par "réseaux sémantiques", "frames", "langages orientés objets" (LOO) et "langages acteurs" ou "RCO" (représentation centrée objet) (56).

2.2.3.1 - Les réseaux sémantiques

Première forme de représentation, ils consistent à représenter l'information sous forme d'un ensemble de noeuds qui représentent les objets et les concepts, et un ensemble d'arcs qui définissent les relations entre noeuds. Cette représentation utilise la notion d'héritage à partir des types d'objets. Le problème rencontré est le temps nécessaire au calcul des informations ou propriétés lorsque la quantité d'informations est importante.

2.2.3.2 - Les frames

Ils trouvent leur origine avec Minsky et sa tentative de décrire les connaissances de la mémoire humaine sous une forme structurée. La quantité d'information manipulée y est gigantesque, d'ou la nécessité, semble-t-il, de la structurer autour d'unités d'information.

Un schéma est une structure de données qui rallie le descriptif et le procédural dans des relations internes prédéfinies. Un schéma est constitué d'attributs supportant une vision particulière du concept qu'ils représentent. Les systèmes les plus connus sont KRL (24) et SRL (191). Les frames utilisent également l'héritage hiérarchique des réseaux, auquel il convient d'ajouter des mécanismes de

contrôle et de filtrage.

2.2.3.3 - Les langages orientés objet

Comme il a été dit précédemment, ils reposent sur une représentation conçue autour de l'entité "objet". Les contrôles se font par l'envoi de messages. La génération des objets se fait à l'aide du concept de classe. Le premier développement fait sur ces langages se trouve dans le langage SIMULA (23), suivit de SMALLTALK (68).

SMALLTALK est un environnement graphique interactif de programmation. Il est construit sur le modèle d'objets communicants. Les objets sont modulaires et la complexité du système est réduite par la minimisation de l'interdépendance des éléments du système. Gérer la complexité est une des clefs de l'approche SMALLTALK à la programmation.

Chaque objet est décrit par l'ensemble de ses champs qui constituent les connaissances de l'objet, et un ensemble de méthodes que l'objet est capable d'exécuter. La différence entre "instance" et "modèle" est relativement importante et demeure fort éloignée de celle définie pour les "frames". La création d'un objet se traduit toujours par une correspondance avec un moule défini par la classe dont l'objet découle.

L'objet joue le rôle d'une boîte noire dont l'accès n'est possible qu'à travers les méthodes qu'il peut ou sait reconnaître.

2.2.3.4 - Les langages acteurs

Ici, les programmes sont conçus comme un ensemble d'objets actifs, ou acteurs, qui communiquent par envoi de message. Ils se caractérisent donc par des objets et des transmissions. Le premier langage connu est PLASMA (107).

Dans une représentation centrée objet, l'objet est transparent. L'accès aux propriétés se fait de manière directe. Les objets et classes peuvent se confondre. Le descriptif des objet suit en général le schéma lisp suivant, tel qu'il est présenté dans SHIRKA (132):

```
(schéma (attribut-1 (facette-11 . valeur-11)
                    .....
                    (facette-n1 . valeur-n1))
        .....
        (attribut-m (facette-1m . valeur-1m)
                    .....
                    (facette-nm . valeur-nm)))
```

Les facettes peuvent être de typage, de valeur, réflexe avec test d'une action. Elles peuvent être constituées de restriction de type, de contrôle (par test d'une clause).

De plus, comme c'est le cas dans LRO2 (138), elles peuvent être accédées par attachement procédural suivant l'état du champ traité.

L'envoi de message peut être fait à un objet, une liste d'objets de même nom ou de même classe, ou à un objet inconnu. Dans ce dernier cas, LRO2 utilise la notion de bus où transite l'information, avec la notion d'ouverture, de fermeture de bus.

Enfin, une notion intéressante que l'on trouve déjà dans les LOO, est l'existence de facette "démon". Une action est alors effectuée lorsque les conditions correspondantes sont réalisées.

En conclusion, ces approches peuvent être reprises pour réaliser le coeur d'un système intégré de CAO-VLSI, à condition de définir exactement ce que l'on souhaite réaliser et la méthode choisie, ce qui va être discuté maintenant.

2.3 - Synthèse

2.3.1 - Les objets

Ne sont manipulés que des objets ayant des champs attributs. Ces objets sont décrits par des modèles que l'on instancie. Une instance est l'image d'un modèle. Les valeurs associées à ces champs sont, soit spécifiées en conformité avec le modèle, soit acquises par défaut, par la description du modèle.

2.3.2 - Les classes d'objets

Un objet peut être décrit sous forme d'un schéma. Ce dernier correspond alors à une classe d'objets. Tout objet est un schéma instance de schéma. Il hérite de la description du schéma de classe, mais précise la valeur de ses attributs. Il est une spécialisation.

2.3.3 - La programmation objet

L'objet est conçu afin que la programmation soit dirigée par les données. Les procédures d'exploitation des données et les données elles-mêmes sont réunies en une seule entité autour de l'objet. Ce dernier est une machine virtuelle capable de répondre à des messages par les méthodes qu'il possède.

Tout objet est instance d'une famille ou classe d'objets. Chaque classe est définie par son nom, ses attributs et les diverses connaissances associées.

Les classes peuvent avoir une déclaration hiérarchique. Une sous classe hérite alors des propriétés de la classe mère.

2.3.3.1 - L'attachement procédural

Aux attributs d'une classe peuvent être associés un certain nombre de procédures, celles-ci étant utilisées lors de l'accès aux valeurs associées à l'attribut, ainsi que lors des opérations d'édition par modification ou suppression.
C'est cette possiblité d'attachement procédural qui permet de réduire la distinction qui existe entre une représentation déclarative et une représentation procédurale des connaissances.

2.3.3.2 - Le message

La communication entre objets se fait par envoi de messages. C'est l'objet destinataire du message, exécutant la méthode correspondante, qui interprète le message de manière adéquate, suivant son contexte.

La seule structure de contrôle est l'envoi de messages.

Un message se caractérise par son identificateur, ses paramètres, et l'objet auquel il est envoyé, lequel recherchera la méthode correspondante et l'exécutera. Lorsque l'objet auquel est destiné un message est inconnu au moment de l'envoi, est utilisé le concept de bus, comme C.Roche le définit dans LRO2(137).

2.3.3.3 - Le choix des objets

La dernière règle considérée est, comme le disent Adèle Goldberg et David Robson dans Smalltalk-80 (68), le choix des objets manipulés. Ce choix est la première étape, et demeure une étape clef. Il n'y a pas de règles, si ce n'est que la puissance d'un système se juge à sa capacité à supporter l'ajout ou la suppression d'opérations.

3 - Spécification du coeur d'un système intégré

Il ne s'agit pas de décrire ici l'ensemble de la base de données sur laquelle peut prétendre reposer un système convenable, ce travail est d'une autre dimension. Il s'agit simplement de présenter un noyau de spécifications de "LA BASE D'OBJETS" sur lequel les processeurs différents que constituent un environnement CAO-VLSI peuvent venir s'appuyer.

Pour les aspects de sauvegarde et de gestion des données, une solution triviale mais efficace est proposée. Cependant elle ne semble pas devoir dépasser le stade de maquette, d'environnement de recherche ou d'environnement universitaire, car tous les mécanismes nécessaires à un

environnement industriel (fiabilité, gestion des versions, travail multi-utilisateurs, etc...) n'ont pas été étudiés. Tous ces aspects sont aujourd'hui fortement développés dans le cadre de bases de données orientées objet OCT (33), DPL (16).

3.1 - Aperçu des bases de données CAO-VLSI

L'utilisation d'une base de données comme noyau d'un système VLSI existe depuis longtemps (58).

3.1.1 - Approche traditionnelle

Les anciennes techniques basées sur le modèle relationnel (3) et le modèle réseau (193) n'ont pas apporté une solution satisfaisante au problème du VLSI (108). R.H.Katz en a donné les raisons suivantes:

-les systèmes classiques, s'ils manipulent bien les données structurées, ont beaucoup de mal à manipuler des textes et images graphiques.

-Les systèmes relationnels supportent mal la notion de hiérarchie.

-la gestion des versions et le partage des espaces mémoire ne sont pas satisfaisants.

-la génération de circuits, traduite par des transactions assez longues, s'adapte mal aux approches traditionnelles plus performantes lors de transactions courtes (13).

3.1.2 - Approche nouvelle

R.H.Katz donne comme conditions à remplir par une base de données VLSI (85), sachant que la conception est obligatoirement structurée, hiérarchisée, et orientée objet:

-la base de données doit permettre la représentation hiérarchique des circuits,

-les différents niveaux (comportemental, fonctionnel, physique) d'un circuit doivent être exprimables,

-la cohérence de l'ensemble du système doit être assurée, en particulier en ce qui concerne les diverses représentations d'un même objet, ou les versions successives,

-le système doit être capable de fournir aux outils les données déja présentes dans la base, et insérées par d'autres outils ou déductibles de données existantes,

-Enfin, il doit être capable de fournir une interface permettant l'insertion de nouveaux outils dans la base de données, ce qui n'est pas le moindre problème.

3.1.3 - Structure de la base de données

Une vision hiérarchique du système est vue, avec au sommet les interpréteurs (interface de manipulation) situés sur le système de gestion du projet, lequel repose sur les objets et leur manipulation.

Ce type de représentation a été développé par R.H.Katz (86) pour la réalisation d'un assembleur de cellules, dans la station de travail CAE 2000 de TEKTRONIX (192), et présentée par W.Kim et D.S.Battory (18,19). Il est également utilisé à Caltech avec SCALE-OCT (33) ou au MIT avec DPL (16).

3.1.3.1 - Spécialisation de la base

Il apparaît du travail de TEKTRONIX (192), que la base de donnée ne peut être indépendante du domaine d'application. Elle se doit d'être spécialisée pour une application donnée, à savoir la conception VLSI, et ceci dès la couche de représentation objet.

3.1.3.2 - Définition des couches

La base de données a l'aspect suivant (108):

* **Interface Utilisateur** (logiciels utilisateur)
  (code d'applications)

* **Interface Centrée Objet**
  (système de fichiers virtuels)

* **Zone de Pagination** (logiciels sur la base)

* **Répertoire**
  (O/S)

* **Disque**

3.1.4 - Bilan de cette approche

Cependant, il convient de dire que cette approche ne connait pas encore de réalisation de caractère industriel sur laquelle il soit possible d'étudier les concepts admis et l'efficacité à résoudre les problèmes posés par la conception de circuits VLSI. L'hypothèse est faite que c'est la voie vers laquelle il convient de se diriger.

La spécification est limitée à une base d'objets, et à un mécanisme de sauvegarde en utilisant pour support le système d'exploitation de la machine utilisée, et sa capacité à gérér des listes de fichiers quand aux problèmes rencontrés (versions, conflits d'édition, droits de lecture écriture, etc...).

3.2 - Base d'objets constituant le noyau du système

Une solution basée sur une représentation centrée objet est proposée, le noyau du système étant constitué d'un squelette minimal de données, indépendant des divers processeurs de construction, de validation, ou de test d'un circuit intégré.

A partir des considérations qui tendent à prouver qu'un circuit peut, et doit être décrit, de façon hiérarchique et structurée, la "cellule" est considérée comme l'objet de base de tous les outils développés, chaque outil se traduisant par une ou plusieurs facettes de cette cellule.

La définition d'ue cellule correspond, soit à un descriptif comportemental, soit à un descriptif structurel, soit à un descriptif physique. Par l'utilisation d'une représentation

"stick", la description physique et le graphe électrique de la cellule deviennent confondus.

Les lieux, moyens d'accès à la cellule dans une démarche hiérarchique "Top-Down", sont les points de connexion, et la représentation hiérarchique se fait par instanciation de cellules sous forme d'appels.

3.2.1 - La cellule

La cellule se compose d'une partie commune à tous les processeurs et comprenant le lieu d'accueil de ces processeurs, ceci pour les rendre communicants, tout en leur assurant une certaine indépendance.

3.2.1.1 - Structure d'une cellule

La structure d'une cellule est la suivante:

-un contexte de déclaration dans la base d'objets,

-une liste de cellules locales (structuration),

-une étiquette de nommage dans la base de données,

-un état, constitué pour chaque processeur exécuté de la date du dernier traitement et des paramètres associés (L'état sera décomposé en deux champs, la date d'édition (version) et les paramètres (flags)),

-une liste d'instances de cellules, ou du moyen d'y accéder dépendant du processeur nécessitant cette instance,

-pour chaque processeur, d'un corps, correspondant à la structure de données ou aux paramètres associés,

-enfin, d'une liste d'attributs extensible pour des applications dynamiques, ou dépendant d'un contexte donné.

3.2.1.2 - Mécanismes associés

La puissance du système doit reposer sur sa simplicité, ses possibilités d'extension, et sa cohérence, d'où la représentation suivante:

-un seul type d'objets structurés (la cellule),

-l'utilisation de mécanismes et de sémantiques de nommage (structuration et construction algorithmique des noms),

-un mécanisme et des sémantiques de filtrage par clef ou par typage,

-La liste des processus exécutés, cette liste se décomposant en trois parties, la version, le corps, et les flags liés à une cellule,

-une liste d'attributs propres à un objet, et pouvant dépendre de son contexte,

-des mécanismes d'accès et de sauvegarde des données.

3.2.1.3 - Schéma d'une cellule

Une cellule se traduit par le schéma lisp:

```
(cellule (versions (p1 . v1) ... (pn . vn))
         (corps    (p1 . c1) ... (pn . cn))
         (flags    (p1 . f1) ... (pn . fn))
         (cellules ...)  ; cellules filles
         (appels ...) ; liste des appels
         (attributs ...) ; liste des attributs
```

"pi" signifie processeur "i", "vi" version du processeur pi, "ci" corps lié au processeur pi, et "fi" flags liés au processeur pi.

Les attributs, comme les autres champs sont des A-listes lisp, les clefs étant les propriétés de la cellule, par exemple le nom de celle-ci: (id . valeur).

3.2.2 - **Environnement d'une cellule**

Un circuit se caractérise par une ou plusieurs hiérarchies. L'environnement d'une cellule donnée est donc un "**espace**" (informations générales, technologie, package, etc...) , une "**hiérarchie**" et une "**cellule mère**".

L'espace de travail est consistué d'une liste de "**répertoires**" où l'utilisateur peut travailler, et d'une liste de "**bibliothèques**" qu'il peut accéder en lecture.

Le choix d'un environnement de travail se traduit donc par la sélection d'un répertoire courant, et de la sélection

d'une hiérarchie courante.

Un utilisateur possède un fichier (similaire à un ".profile" sous UNIX (171)) dans lequel est décrit un certain nombre de paramètres, parmi lesquels les environnements de travail, la technologie employée, les routeurs par défaut, le terminal utilisé, le nommage par défaut, la liste des outils utilisés, le mode d'affichage choisi, la profondeur d'affichage d'une hiérarchie par défaut.

Chaque cellule peut être accédée, une fois un environnement défini, à partir de son nom et de l'espace qui lui est associé.

3.2.3 - Sauvegarde, restauration des données

Il a été vu que dans un espace de travail donné, l'utilisateur est amené à traiter une liste de hiérarchies.

Chaque hiérarchie se traduit par un squelette qui est indépendant de l'outil utilisé, contenant l'arbre de définitions, l'arbre des appels, et un certain nombre d'arguments. Parmi ces arguments, il y a, pour chaque cellule, un identificateur unique dans l'espace donné. A partir de cet identificateur, le corps de chaque cellule, pour chaque processeur, est sauvegardé séparement sous forme d'un fichier.

3.2.3.1 - Sauvegarde

La sauvegarde se caractérise par:

-la sauvegarde des corps de cellules sous forme de fichiers,

-la sauvegarde du squelette de la hiérarchie.

L'intérêt de cette démarche est de disposer ainsi des moyens de contrôle sur la structure de donnée hiérarchique, et des moyens de parcours des hiérarchies, sans avoir à manipuler l'information, pour chaque cellule, correspondant à chaque processeur utilisé.

La sauvegarde d'une hiérarchie se fait en deux parties:

-le test de tous les noeuds ayant été édités et la sauvegarde de leur contenu,

-la sauvegarde du squelette de la hiérarchie.

Les tests d'édition se font sur les champs flags. La sauvegarde se traduit par l'envoi du message "sauvegarde" à la hiérarchie traitée. Celle-ci le traite en envoyant le message à chaque cellule. Une cellule exécute la sauvegarde en faisant un accès en mode sauvegarde, au champ corps qui lui est associé. La procédure d'accès à ce champ est alors lancée. Elle effectue la sauvegarde du corps si il y a eu édition.

3.2.3.2 - Restauration

La restauration d'une hiérarchie suit un procédé analogue. L'utilisateur spécifie la hiérarchie qu'il souhaite traiter. Le squelette de celle-ci est alors chargé (opération extrèmement rapide). Puis, lors de l'accès aux corps des cellules, en fonction de l'outil employé, ces derniers sont

chargés automatiquement. De même, lors de parcours hiérarchiques des arbres d'appels, des hiérarchies non présentes en mémoire peuvent être chargées automatiquement.

Le problème de "sauvegarde/restauration" devient transparent à l'utilisateur. Le chargement des hiérarchies est extrêmement rapide, et seules les informations utilisées sont chargées, d'ou un gain important sur le temps de chargement de l'ensemble des informations, les entrées-sorties étant incontestablement le point faible de la plupart des systèmes.

3.2.3.3 - Réalisation

La "sauvegarde/restauration" se traduit par un seul type de commande:

(action objet-traité environnement paramètres)

avec:

action: sauve/restaure
objet-traité: la hiérarchie traitée, la cellule traitée.
environnement: répertoire de sauvegarde
paramètres: liste de paramètres de sauvegarde.

Le calcul du répertoire de sauvegarde est effectué dynamiquement en suivant le principe d'accès aux noms par typage.
Le calcul d'un nom de fichier a toujours la forme:
-prefix (lié au répertoire de sauvegarde)
-nom (lié à l'objet)
-suffix (lié au processeur)

Le nom est calculé par concaténation des trois champs, avec des variations suivant le système d'exploitation.

3.2.3.4 - Analyse

Cette méthode de "sauvegarde/restauration" est assez efficace et performante quand à l'accès aux objets, ainsi qu'au portage d'une machine à l'autre, puisque la seule entité réellement manipulée est le fichier. Son principal inconvénient est, pour une hiérarchie, la mise à plat de l'ensemble des fichiers correspondant à l'ensemble des processeurs exécutés sur chaque noeud. Il en découle un nombre de fichiers important. Une démarche intéressante serait de regrouper toutes les informations concernant une cellule dans un même fichier avec des clefs d'accès et des mécanismes d'édition par tranche de fichier, ce qui est relativement facile à faire, et ne pénalise le système qu'à l'accès aux informations sur un corps de cellule, ce qui en principe est négligeable à coté de l'information qui est chargée. Une autre approche est d'associer un fichier à chaque processeur. Il y retrouve l'information par des accès séquentiels. Cette approche est simple et réduit considérablement le nombre de fichiers à gérer. Les problèmes rencontrés sont la taille des fichiers et le temps d'accès à une donnée.

3.2.3.5 - Problèmes de fiabilité

Un autre problème rencontré est l'absence de fiabilité du système. En effet, lors de la perte du fichier contenant le squelette d'une hiérarchie, ou lors de la perte de cohérence du fichier, suite à un problème système, il n'est plus

possible de retrouver la liste des fichiers correspondant aux processeurs exécutés sur le corps des cellules de la hiérarchie. Il reste un ensemble de fichiers inutilisables. Soit une autre solution doit être envisagée, soit un mécanisme de restauration d'une hiérarchie perdue, à partir des fichiers correspondant au corps des cellules, doit être réalisé. Cette dernière solution semble intéressante à implanter.

3.2.3.6 - Gestion des versions

Enfin, la gestion des versions ne semble pas être un problème primordial. Si un concepteur veut mémoriser à un moment donné son travail, il a toujours la possibilité de le faire par réalisation d'une copie de la hiérarchie traitée. Le système doit pour cela comporter la commande "dupliquer".

S'il est nécessaire de pouvoir faire des retours en arrière sur un circuit, une approche identique à celle proposée dans le système d'exploitation VMS (176) peut être considérée. La numérotation des versions se fait par existence d'un champ indice dans le nom des cellules. A chaque édition, l'incrément est fait et une nouvelle hiérarchie est sauvegardée, ce qui revient à faire une copie avec renommage automatique.

Les approches qui consistent à mémoriser les différentiels entre versions, sur la même structure, ou à associer toutes les versions sur la même hiérarchie alourdissent considérablement tous les mécanismes et amènent à des traitements ou des décisions extrêmement complexes lors de l'édition (suppression de branches) de la structure de la

hiérarchie traitée et ne semblent pas être le point essentiel quand à la qualité d'une station de travail.

3.2.4 - Structuration des noms

La structuration des noms est incontestablement le moyen de concevoir des outils puissants, aussi bien au niveau de la génération algorithmique d'un circuit, qu'au niveau validation, test. C'est le moyen d'accéder à des objets par des particularités qui lui sont propres, et bien souvent contenues dans le nom (exemple: une UAL 32 bits se voit souvent attribuée le nom UAL-32), il est donc intéressant de pouvoir bénéficier de ces informations, de concevoir des noms de façon algorithmique lors de la conception procédurale de circuits. C'est un moyen de nommage automatique qui peut s'avérer très performant.

Les noms sont vus sous forme d'un objet "obj-nom". Cet objet est typé, on y accède par sémantique. Il existe un certain nombre de types de base, dont la "string", le nom de fichier, etc....

Au niveau du noyau, tout nom accédé par procédure doit retourner une chaîne de caractères composée des caractères alphanumériques admis pour un symbole Le_Lisp (43).

L'ensemble du noyau n'utilise que deux primitives autour des noms, la primitive de comparaison de noms, faite sur les chaînes de caractères (au lieu de l'objet obj-nom), ce qui rallonge le traitement, mais le rend plus général, et la primitive d'affectation ou de retrait d'un nom.

(donne-nom objet cellule contexte . valeur)

-objet est l'objet auquel on demande le nom,
-cellule est la cellule dans laquelle se trouve l'objet,
-contexte est le contexte hiérarchique d'action, il permet entre autre de choisir le constructeur du nom, par le choix d'un package,
-valeur est () en lecture, le nom affecté à l'objet en écriture.

Le nom d'un objet, quelque soit l'objet traité (la cellule, l'appel, le point de connexion, etc...) est toujours positionné dans la liste des attributs de l'objet, sous la clef "ID".

La lecture d'un nom se fait par le lancement de la procédure associée au champ de l'objet traité. Cette procédure se caractérise par le type de l'objet "obj-nom".

Afin d'accéder rapidement à une cellule à partir de son nom, à chaque symbole lisp correspondant est associé, dans sa P-liste, un double pointeur sur la hiérarchie et la cellule concernée. Ces pointeurs sont initialisés lors du chargement en mémoire des cellules.

3.2.5 - Opérations générales sur une hiérarchie

Les opérations principales sont, la génération, la suppression et l'édition.
Un objet est toujours considéré dans son environnement (contexte + cellule de déclaration). Toute action sur un objet a donc la syntaxe suivante:

(action objet contexte cellule paramètres).

3.2.5.1 - Accès à une cellule

A partir d'une cellule, l'accès à toute cellule suit une règle de visibilité hiérarchique montante (type algol). De ce fait, toute cellule comporte un nom unique dans son contexte local. La recherche d'une cellule se fait par la fonction:

(donne-cell cellule environnement "nom")

qui retourne soit la cellule, soit ().

La cellule est l'objet traité, et l'environnement est constitué du contexte et de la cellule mère.

3.2.5.2 - Edition des hiérarchies

Deux opérations sont définies, celle d'ajout d'un élément à une hiérarchie, et celle de retrait.

Une cellule ne peut être retirée si elle possède des définitions locales. Lorsqu'une définition de cellule est retirée, en raison de la visibilité montante, une incohérence peut être créée dans la base de données. De ce fait, lors du retrait d'une cellule, il y a parcours hiérarchique, à partir de la cellule mère de la cellule retirée, pour voir si une cellule locale instancie la cellule retirée. Dans ce dernier cas, un message lui est envoyé. Lors du traitement suivant, la cellule disparue

n'est pas recherchée, à moins qu'entre temps une autre cellule de même nom soit déclarée dans l'espace de visibilité. Dans ce cas, il convient de n'effectuer qu'une simple remise à jour.

L'opération d'ajout d'une cellule dans la base se fait après envoi du message à la cellule éditée pour savoir si cet ajout est possible. Il l'est lorsque l'arbre de visibilité existant n'est pas modifié, dans le cas contraire, il y a échec et l'opération est refusée.

3.2.6 - Environnement graphique

Il apparaît que les problèmes d'ergonomie et de fonctionnalité ont un rôle considérable quand à la qualité d'un outil de CAO.
L'expérience montre qu'un outil de faible capacité, avec une interface humaine bien conçue, est de loin préféré par le concepteur à un outil puissant dont l'interface est mauvaise.

D'autre part, il est nécessaire que tous les outils travaillant dans une station puissent le faire sans avoir leur interface respectif. A ce sujet, l'approche de SMALLTALK, "environnement graphique interactif de programmation" (68) est extrêmement intéressante, dans la mesure où l'aspect graphique est traité comme une entité à part entière.

3.2.6.1 - Traitement du problème graphique

Il est nécessaire de traiter l'aspect graphique d'une station de travail comme un problème indépendant des

processeurs, chaque processeur pouvant être amené à communiquer avec le monde graphique via l'envoi de messages.

Dans les approches traditionnelles, il est admis que l'on prend pour éléments de base du traitement graphique un ensemble de fonctions parfaitement décrites (GKS, PLOT-10, ...). Cependant, il apparaît que ces fonctions ne sont pas d'un niveau suffisant pour satisfaire les besoins rencontrés, et surtout, elles ne présentent aucune structuration. La conception d'un environnement graphique doit donc se faire par un interface de plus haut niveau, un interface parfaitement structuré et qui puisse fonctionner comme un ensemble d'objets dans une représentation centrée objet. Ceci sous entend également que le type de terminal sur lequel on travaille admet un certain niveau de performance et de complexité (exemple: gérér les coordonnées utilisateur pour un "layout"). Les expériences faites avec du matériel beaucoup moins performant montrent que les problèmes rencontrés au niveau de la gestion du graphique entraînent la réalisation de mauvais concepts au niveau des actions effectuées par les processeurs. Il leur est assez difficile et coûteux d'effectuer ce qu'un terminal de haut niveau sait faire.

3.2.6.2 - Terminal de haut niveau

Un terminal de haut niveau se caractérise par la possibilité de déclarer des vues, des sous-vues, avec un isomorphisme entre l'écran, la tablette graphique ou la souris, et la structure de données manipulée. L'utilisation de segments et les possibilités de hiérarchisation des segments, la maîtrise des plans et couleurs associées, la possibilité de

définir ses propres motifs, l'existance d'un traitement alphanumérique puissant et aux normes connues (vt100 ou autre) sont nécessaires.

Tout comme l'est le fait de pouvoir laisser le système graphique gérer les coordonnées. Les processeurs n'ont plus alors qu'à se préoccuper des coordonnées réelles utilisées. Les coordonnées graphiques et les transformations qui en dépendent sont gérées au niveau le plus bas, à savoir par le terminal.

3.2.6.3 - Processeur graphique

L'environnement graphique est vu comme une base d'objets constituée à partir de la liste des processus utilisés.

Un écran peut être considéré comme une somme de vues, chaque vue se décomposant en sous-vues. Une vue peut être visible, il lui est alors associé un mode d'affichage, ou non-visible. Chaque vue peut être associée à un processeur. Chaque élément dessiné est associé à un objet de la base d'objets manipulés. Ainsi, une correspondance biunivoque est réalisée. Chaque action au niveau du graphique se traduit par l'envoi du message associé au processeur en cours. Il est reçu par la liste des objets en correspondance biunivoque avec les objets de la représentation graphique. A chaque action au niveau de la structure d'objets manipulés, la répercussion au niveau de la structure graphique correspondante est assurée par l'opération réciproque.

3.2.6.4 - Intérêt du processeur graphique

Cette approche permet de travailler au niveau des processeurs avec dans l'esprit des concepts graphiques de haut niveau. L'intégration est très rapide, puisque les seules procédures à écrire sont celles décrivant le graphique associé aux objets manipulés par un processeur, l'ensemble des concepts graphiques étant gérés par l'interface graphique.

C'est l'approche que prennent de nombreux systèmes, où le noyau graphique est une entité à part entière, et où les processeurs ne travaillent que sur un environnement graphique "virtuel". Le portage d'une machine à l'autre est alors facilité, à condition que les machines soient capables de supporter toutes les fonctions nécessaires, ou capables de les simuler.

Le choix de supports graphiques est encore limité, cependant le développement de cartes graphiques très performantes devrait permettre dans les années à venir de bénéficier d'un environnement matériel de qualité à bon marché. Le problème n'en reste pas moins la spécification et la réalisation d'un noyau graphique capable de supporter toutes les applications que les divers processeurs utilisés peuvent exiger.

3.2.7 Ajout de processeurs au noyau

Il convient de voir comment se comporte le noyau spécifié précédemment, lorsque de nouveaux outils sont ajoutés à la base, l'extensibilité étant une chose vitale pour un système qui ne peut en aucun cas être fermé.

3.2.7.1 - Caractéristiques d'un processeur

Chaque processeur est caractérisé par son corps, ses flags son état et la liste de procédures qui y sont attachées. Vouloir insérer, ou retirer, un processeur, revient à déterminer les interfaces existants entre tous les processeurs.

3.2.7.2 - Processeur graphique

En ce qui concerne le processeur graphique, il ne doit pas y avoir de problèmes, car en principe il est conçu suffisamment général pour accepter de dialoguer avec n'importe quel processeur. Dans le cas contraire, il conviendrait de l'améliorer.

3.2.7.3 - Interdépendance de processeurs

Il est un certain nombre de processeurs qui peuvent avoir un degré de dépendance. Par exemple, la recherche de cohérence au niveau fonctionnel peut se traduire par une recherche de cohérence au niveau topologique. L'algorithme de cohérence travaille avec plusieurs types de représentations.

Il apparaît donc une chose: toutes les actions, qui pour un processeur existant vont entraîner des actions sur un nouveau processeur lors de son introduction dans la base, doivent être traduites par de nouvelles règles. Ceci peut sans doute être fait de façon simple dans le cadre d'une programmation prolog. Cependant, en raison de la difficulté rencontrée à définir l'ensemble des besoins associés à chaque outil, cette approche n'est pas abordée. L'ensemble des procédures liées à un processeur et qui sont amenées à

dialoguer avec d'autres processeurs doivent être isolées, et revues lors de l'ajout d'un outil.

Ceci nécessite la connaissance du fonctionnement du coeur du système, la connaissance des fonctions interfaces de chaque outil, et la connaissance des procédures présentant une interaction ou caractérisant un coefficient de corrélation entre outils.
Par exemple, si on insère un générateur automatique de "layout juste par construction" dans le système, la règle selon laquelle le vérificateur hiérarchique de règles de dessin n'a pas à vérifier le coeur des cellules générées, par cet outil, doit être insérée dans l'algorithme de vérification.

L'ensemble des procédures présentant une dépendance vis à vis des processeurs autre que celui auquel il est attaché demeure cependant réduit et concerne surtout la cohérence de la base de données. Les données manipulées par un processeur à un instant donné correspondent-elles bien à celles manipulées à un autre instant par un autre processeur, ou du moins ne sont-elles pas contradictoires, telle est la question posée par les algorithmes de recherche d'incohérence.

3.2.8 Gestion de la cohérence des données

C'est incontestablement le problème rencontré par la plupart des systèmes de CAO-VLSI. En raison des différentes descriptions et des spécifications possibles pour un circuit à chaque niveau, il est difficile de maintenir l'ensemble de ces données cohérentes.

3.2.8.1 - Choix d'une méthodologie

Les approches relationnelles et traditionnelles connaissent ici bien des difficultés, alors que dans une approche objet, une solution simple existe. Elle consiste à vérifier, par un attachement procédural, lors de l'accès au champ "corps" d'une cellule, pour un processeur donné, la cohérence des données dans une démarche "Top-Down".

Ce qui revient à dire que la seule cohérence qui doit être vérifiée dans une base de représentation d'un circuit, est celle vis à vis des éléments que l'on manipule. C'est lors de la manipulation des objets que le contrôle de la cohérence est effectuée. Autrement dit, il n'a pas de "back-tracking" ou chainage arrière sur les arbres d'appels et de définitions.

3.2.8.2 - Outils de gestion de la cohérence

En mémoire, à tout instant la liste des squelettes des hiérarchies utilisées est disponible (un système de pagination extrêmement simple peut être réalisé afin de traiter le cas d'une limitation sérieuse de l'espace mémoire, ou de la manipulation de grosses hiérarchies).

Chaque noeud d'un arbre comporte la liste des processeurs utilisés, leur état, la date de leur utilisation et le moyen d'accéder aux données.
Il suffit de faire une vérification incrémentale descendante de la cohérence par comparaison de dates sur l'arbre d'appels.
La cohérence d'une cellule vis à vis d'un processeur peut se

traduire par deux possibilités:

-le processeur est indépendant. Il a sa validation propre. Il ne va pas lancer d'autres processeurs. Il n'effectue qu'un simple parcours hiérarchique récursif.

-le processeur est corrélé à d'autres. La cohérence est obtenue si la cohérence propre du processeur est vérifiée, et si à l'envoi du message "cohérence?" aux processeurs concernés correspond une réponse favorable.

3.2.8.3 - Intérêt pour la conception

Le choix de ce mécanisme de cohérence présente un grand intérêt, quant à la sûreté de conception. Il n'est pas pénalisant, en raison du type de représentation choisi (squelettes de hiérarchies auxquels sont attachés les paramètres minimum), et du langage utilisé (Le_Lisp) particulièrement bien adapté aux parcours récursifs de hiérarchies et de listes.

Le seul problème rencontré est le choix de l'action à effectuer lors de la détection d'un incohérence vis à vis d'un processeur. La décision à prendre est alors prise au niveau du processeur.

Le cas de l'incohérence de l'arbre d'appel sera traité pour la représentation physique du circuit, en fonction du type de génération (édition incrémentale, construction algorithmique, génération par procédure, génération par outil spécialisé, ...) dans le chapitre suivant.

TROISIEME PARTIE

OUTILS DE GENERATION DE MASQUES

1 - Vue d'ensemble

Un des buts recherché est de trouver les outils qui permettent de concevoir des circuits intégrés de grande densité, les problèmes rencontrés étant: le temps de conception (choix d'une architecture, décomposition, simulation, dessin des masques), la validation du circuit obtenu et les tests de fonctionnement.
Sont étudiés les outils touchant les masques, vus dans un contexte plus global.

1.1 - Noyau des outils de conception

Dans le chapitre précédent un noyau de base à partir duquel les processeurs sont lancés a été décrit. Ce noyau contrôle les accès aux données, leur validation et leur cohérence, un mécanisme de nommage général et puissant. A ceci il convient d'ajouter les mécanismes de filtrage des données par attributs que l'on rencontre dans une représentation objet.

1.2 - Outils de génération des masques

Au niveau du dessin des circuits, quatre composantes apparaissent: les générateurs, les éditeurs, les compilateurs et les routeurs globaux.

1.2.1 - Structuration des circuits

Si pour point de départ une méthodologie descendante et hiérarchique est prise, le problème se décompose en une description du circuit à l'aide d'un langage de haut niveau qui va le structurer en blocs fonctionnels, et la génération de ces blocs.

Le problème rencontré lors de la manipulation des blocs, est la réalisation du plan de masse et le routage global du circuit. Ceci est un sujet pour lequel de nombreuses solutions existent, basées la plupart sur la résolution d'un graphe des contraintes (47,134,179,189).

1.2.2 - Description des blocs

L'action essentielle retenue est la nécessité de pouvoir spécifier un certain nombre de données lors de la description des blocs (nombre de connexions, position, ...). Ces données doivent être prises en compte par les générateurs de blocs, l'approche la plus classique étant la réalisation de ces blocs par assemblage de cellules "stretchables".

La génération des blocs se fait soit de manière purement manuelle, soit par compilation d'une description de haut niveau (PAOLA (49)), soit par exécution d'un algorithme d'assemblage de cellules feuilles (LUBRICK (146)), soit par exécution de procédures (ICPL (93)).

1.2.3 - Cohérence des différentes descriptions

Les problèmes rencontrés au niveau de la génération des

masques sont le rapport entre les données générés et les autres outils:

-génération de cellules.

-structuration des circuits.

-cohérence de l'ensemble des données (vrai pour tous les processeurs).

-volume de données à manipuler et vitesse d'execution des processeurs utilisés.

La vision d'une station de travail CAO-VLSI peut être que la réalisation d'un circuit intégré se traduit par un ensemble de programmes qui, chacun à son niveau décrit le circuit réalisé. Le problème est alors de contrôler que les diverses représentations traduisent le même circuit, et de faire passer les informations ou les données générées, d'un niveau de description à l'autre.

1.2.4 - Approche symbolique

En étudiant l'aspect purement graphique de la chose, il apparaît qu'une approche purement symbolique est extrêmement séduisante, cependant les différentes méthodes de génération d'un bloc montrent certaines limites, en particulier au niveau des générateurs automatiques.

1.2.4.1 - Générateurs et langage procédural

Le passage de générateurs de layout aux masques se fait par l'intermédiaire d'interfaces. L'utilisation d'une représentation symbolique permet aux générateurs de devenir beaucoup plus puissants, plus simples, les problèmes rencontrés étant le temps d'exécution et le volume des

données (Ex: Le pla , la RAM, la ROM, le canal de routage).
Le générateur le plus puissant est celui qui manipule une représentation symbolique, construit une structure symbolique et en déduit les masques correspondants par une loi fonction de la position du symbole dans la structure (exemple PAOLA (129)). Le temps de réalisation et le temps de mise au point du générateur sont relativement long (plusieurs années pour PAOLA).
Le même générateur réalisé avec un langage procédural (DPL (16), ICPL (93), ...) est beaucoup plus élégant, les vérifications électriques et de règles de dessin, quasi-incrémentales, rendent le temps de réalisation et de mise au point du générateur assez faible. Cependant, lorsque l'on travaille à un haut niveau, la quantité d'information générée est très importante (manipulation de noms, symboles, graphes, procédures complexes, ...) et les performances diminuent.

La conclusion est que l'utilisation d'un langage symbolique de haut niveau permet de manipuler des concepts et simplifie la génération du layout par l'écriture rapide de générateurs. Il est possible d'écrire rapidement le générateur correspondant au besoin rencontré. Le générateur peut être "intelligent" (prise de décision ... générateurs lisp/prolog). Cependant elle n'est pas la meilleure solution pour les générateurs de blocs simples ou réguliers (ROM, RAM, PLA).
En conséquence, il faut à la fois travailler au niveau d'un langage symbolique, et savoir faire appel à des générateurs spécialisés. Par exemple, pour la génération de canaux de routage, on mémorise les points d'interconnexions, le nom du routeur et ses paramètres. Il doit alors exister un certain

nombre de procédures correspondant à l'exécution du canal dans différents contextes (exécution pour visualisation, matérialisation, extraction des frontières du canal, etc ...). Les outils travaillant sur le symbolique peuvent alors fonctionner sans avoir à se préoccuper de ce qu'est réellement le canal de routage.

1.3 - Environnement et outils de génération des masques

Un environnement de CAO doit être capable d'utiliser des concepts basés sur un éditeur symbolique, mais doit également être capable d'admettre n'importe quel générateur de blocs. Dans le cas d'un contexte unifié, cela sous-entend que chaque outil de génération apporte les informations de caractère hiérarchique que les autres outils peuvent attendre de lui.

2 - Corps d'une cellule

Il a été vu dans le chapitre précédant qu'une cellule avait pour structure:

```
(cellule  (versions  ...)
          (corps     ...)
          (flags     ...)
          (cellules  ...)
          (appels    ...)
          (attributs ...))
```

A partir du moment ou les masques peuvent être générés par plusieurs processeurs différents, il apparaît qu'il faut savoir interpréter quel est le processeur ayant agi et

quelle action exécuter.

2.1 - Génération automatique

Pour un générateur automatique ou un générateur par exécution de procédure, le corps de la cellule ne contient que les informations liées au générateur, soit le nom du générateur, les paramètres d'execution, et éventuellement le lieu de stockage des résultats obtenus. A ceci il convient d'ajouter la vision externe de la cellule, constituée des points de connexion à la cellule, et éventuellement des frontières et ombres définies plus loin.

2.2 - Génération semi-automatique

Pour des générations de cellules par une double action, manuelle et automatique, cas du symbolique virtuel, le problème est différent. Une structure de données, ayant un sens électrique est fournie, en opposition au cas précédent. De ce fait, les processeurs travaillant sur une structure électrique peuvent parfaitement fonctionner. Il convient de mettre dans le corps de la cellule réalisée, la structure de donnée. A cette structure, sont associés les outils ou paramètres de génération des masques, et éventuellement le lieu de stockage des masques générés.

Il est encore plus intéressant de voir un symbolique "mou" comme cible des générateurs symboliques non métriques (exemple: compilation de graphe électrique), un processeur permettant de transformer ce symbolique mou en un symbolique métrique dur. Le processeur symbolique mou est associé soit à un compacteur, soit à des mécanismes de "stretch"

fournissant des cellules décrites en un symbolique métrique.

2.3 - Génération manuelle

Enfin, dans le cadre d'une édition manuelle de cellules, à des fins d'assemblage de cellules de grande densité, une structure correspondant à une représentation symbolique métrique doit être présente. Elle devient également la cible d'une génération par symbolique mou et compactage avec contraintes.

2.4 - Processeurs qui en découlent

En conclusion, trois types de processeurs peuvent être attendus dans la base. Pour le premier type, il convient d'adapter les interfaces (vision externe d'une cellule, fonctionnement des autres outils). Pour les deux autres, ils portent en eux toute l'information électrique associée aux masques, le symbolique dur étant la réalisation d'un symbolique mou associé à un certain nombre de paramètres et de contraintes topologiques.

3 - Symbolique métrique

3.1 - Spécification

3.1.1 - Principe (9-1)

L'outil est conçu comme un outil capable de générer des masques de circuits avec une grande densité. Ces masques

peuvent être obtenus par l'expansion d'un dessin squelettisé métrique planaire, constitué d'articulations, de segments et de zones, qui constituent les éléments de base du circuit.

3.1.2 - Les articulations

Les articulations sont vues comme les discontinuités électriques ou topologiques. Elles peuvent indiquer la présence de contacts, un changement de largeur de connexion pour un niveau technologique donné, les points d'entrée/sortie au niveau des canaux des transistors, et les changements de direction.

3.1.3 - Les segments

Les segments sont vus comme les éléments de continuité du dessin. Ils représentent les conducteurs et les canaux des transistors dans leur partie rectiligne. Les segments reposent sur les articulations qui les délimitent.

3.1.4 - Les zones

Les zones représentent des parties de circuit à caractère équipotentiel, n'ayant pas d'orientation précise, et se comportant comme un noeud du graphe électrique simplifié (présence une capacité dans une approche plus fine).Comme les segments, elles reposent sur un ensemble d'articulations.

3.1.5 - Facette métrique des cellules

Une représentation squelettisée métrique est obtenue, où les

informations électriques et topologiques sont réunies en un graphe. Les éléments ont chacun une expansion qui lui est propre et "invariante". Seules les articulations s'expansent en fonction du contexte. Les interfaces qui permettent la construction hiérarchique de circuits par instanciation de cellules se traduisent par des articulations typées "point de connexion". Ce sont les seuls moyens d'accéder à la cellule.

La facette **"STICK-METRIQUE"** d'une cellule présente alors l'aspect suivant:

-Version : date de génération, auteur, technologie
-Flags : status des erreurs d'édition
-corps : vision externe (points de connexion)
segments
zones
articulations
-appels : référence aux cellules appelées
date d'instanciation
liste des paramètres propres à toutes les instances d'une même cellule
liste des paramètres dépendant de chaque instance.
-attributs : nommage
-cellules : liste des cellules filles

3.2 - Problème technologique

La réalisation d'un outil de génération de circuits a pour principale difficulté la dépendance technologique. Il est nécessaire de pouvoir passer d'une technologie à l'autre.

Car même si on arrive à déterminer des règles standard, l'expérience montre qu'elles sont limitées par les contraintes liées à chaque technologie.

3.2.1 - Base technologique

En créant une base technologique, on doit permettre aux outils travaillant sur les cellules, ou les hiérarchies de cellules de devenir (plus ou moins) indépendants de la technologie, tout au moins pour les grandes familles de technologies. Un changement ne remet alors pas en cause, ni la cohérence de la base de données, ni les algorithmes. Seule l'adaptation des cellules de base est nécessaire.

3.2.2 - Différentes règles

Il est possible de considérer deux classes dans les règles technologiques. Tout d'abord, les règles d'ordre numérique qui servent à quantifier des relations de distance, de surface au niveau des masques. Ces règles peuvent être représentées facilement par des fichiers technologiques, et le sont dans la plupart des systèmes. Ensuite des règles d'ordre structurel ou topologique, qui déterminent des configurations obligatoires ou interdites. Ces règles posent les gros problèmes, car elles varient de façon considérable suivant les technologies (en MOS contact sur grille, ou VIA interdit sur zone active), et les algorithmes de génération d'un symbolique (expanseur, DRC, routeur, ...) en dépendent fortement. Le problème est de trouver le moyen de minimiser le coût logiciel lors du passage d'une technologie à une autre.

3.2.3 - Indépendance technologique

Les règles de dessin introduisent des relations et des contraintes entre niveaux technologiques. Il convient de conçevoir un système où l'on admet l'existance de ces règles, mais où on ne les traite pas directement. La structuration de la programmation doit être faite afin que le changement de technologie n'entraîne qu'un changement de fichiers technologiques.

3.2.3.1 - Symboles et règles technologiques

La solution proposée est de considérer que dans un outil de "layout", à chaque technologie déclarée, on associe un certain nombre de niveaux technologiques (mat1, ... , matn). Les objets manipulés possèdent un certain nombre de propriétés décrites dans un modèle. Parmi ces propriétés se trouve un descriptif des niveaux technologiques utilisés, ainsi que le dessin associé (un rectangle, ou une liste de rectangles). On a ainsi le moyen d'exprimer la représentation de tous les objets manipulés. Se pose alors le problème des règles contextuelles (expansion, violation de gardes), que l'on rencontre essentiellement lors de l'édition (graphique ou algorithmique) d'un layout. La solution qui consiste à déterminer une fenêtre d'édition, propre à chaque objet, a été choisie. Dans cette fenêtre, nous filtrons les objets avec lesquels la fenêtre d'édition présente une intersection. Il convient alors de déterminer l'action à exécuter pour maintenir le graphe cohérent.
Pour cela les objets de la liste obtenue sont typés suivant leur interaction avec l'objet manipulé. Soit ils sont en violation de garde technologique (suivant le fichier technologique) et entraînent une erreur de dessin, soit ils

sont en relation d'édition (ré-expansion des interfaces) ou en simple relation de proximité (par equipotentielle). Une fois ces typages effectués, le graphe est mis à jour, chaque objet ayant une sémantique électrique.

Les algorithmes de typage des objets sont technologie dépendants, ils sont écrits sous forme de règles écrites dans le "package" de la technologie concernée, et associés aux objets manipulés:

(send TECHNO type-edition obj0 obji l-obj cell)

De la même façon les règles de mise à jour du graphe dépendent de la technologie (erreur, édition des interfaces, simple proximité). Les algorithmes associés sont aussi "packagés" par la technologie employée.

3.2.3.2 - Problèmes rencontrés

La solution proposée est assez coûteuse en temps, en raison du nombre d'actions réalisé. Elle est fonction du nombre d'objets dans la cellule traitée et de la complexité des algorithmes de détection, d'expansion, de parcours d'équipotentielles. Cependant, la réalisation d'algorithmes simplifiés et de mécanismes de filtrage des données la rend satisfaisante.

Cette solution ne rend pas l'outil de génération du "layout" indépendant de la technologie. Elle permet simplement de minimiser sa dépendance. Le passage d'une technologie à une autre se fait par la réécriture de ces règles technologiques.

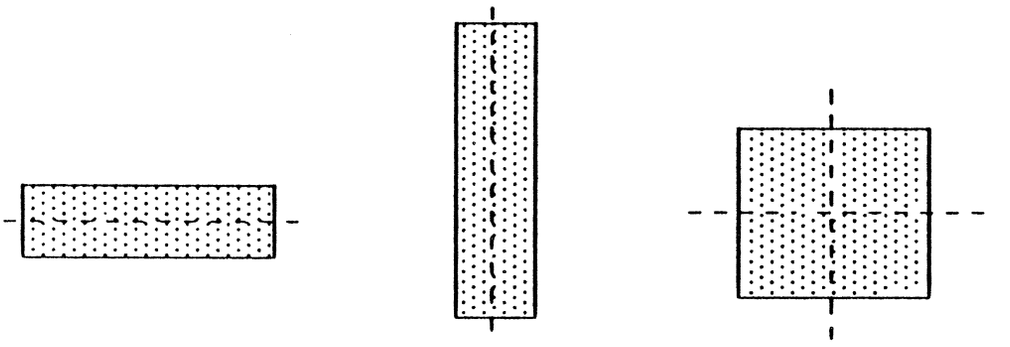
Il n'existe pas, sur le marché, de stations de travail ayant une indépendance technologique parfaite, et pour certains produits (exemple VTI) le passage à une nouvelle technologie s'avère parfois long et coûteux. Avec la solution choisie, il semble que l'adaptation est plus facile. Cependant,une étude approfondie, et le choix de nouveaux critères plus généraux (indépendants des technologies, dépendants des matériaux utilisés) semble nécessaire.

3.3 - Réalisation

Le but premier (9-1) est de concevoir la maquette d'un éditeur symbolique permettant la conception de "layout" aussi dense que peut le faire un éditeur CALMA (38). Il ne s'agit que d'une simple maquette universitaire.

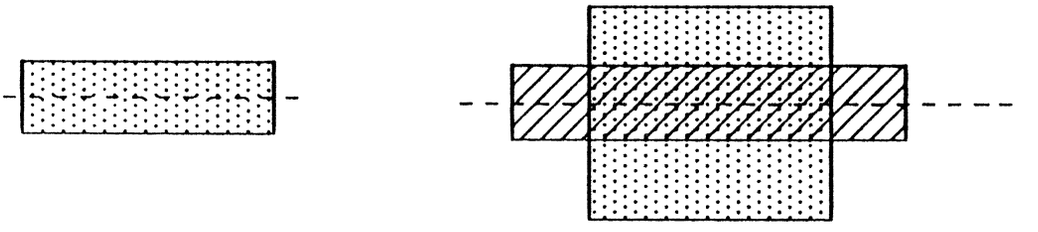
3.3.1 - Choix d'une représentation "manhattan"

Les mécanismes et algorithmes d'édition cellulaire réalisés peuvent, à priori, traiter des symboles sans restriction sur les angles. Cependant, seuls les angles multiples de 15 degrés ont été considérés, les valeurs (0°, 30°, 45°, 60°, 90°) étant largement suffisantes à toute application VLSI.
A l'implantation des algorithmes sur SM90, machine 32 bits (68000), il apparaît une dégradation du temps d'exécution aussi bien au niveau de l'expanseur que du DRC incrémental. C'est pourquoi les valeurs (0°, 45°, 90°) sont choisies. De manière hiérarchique, l'utilisation des angles non manhattan multiplie environ par 100 la complexité des algorithmes sur les frontières et le placement relatif des cellules, aussi ils ne sont pas traités. Néanmoins, l'utilisaton de moyens informatiques plus puissants et la réalisation d'algorithmes plus performants doit permettre l'utilisation de ces valeurs angulaires et l'obtention d'un gain de



segment horizontal segment vertical zone

FIGURE S-1



segment simple
(diffusion)

segment complexe
(transistor)

FIGURE S-2

Expansion simple

Expansion complexe

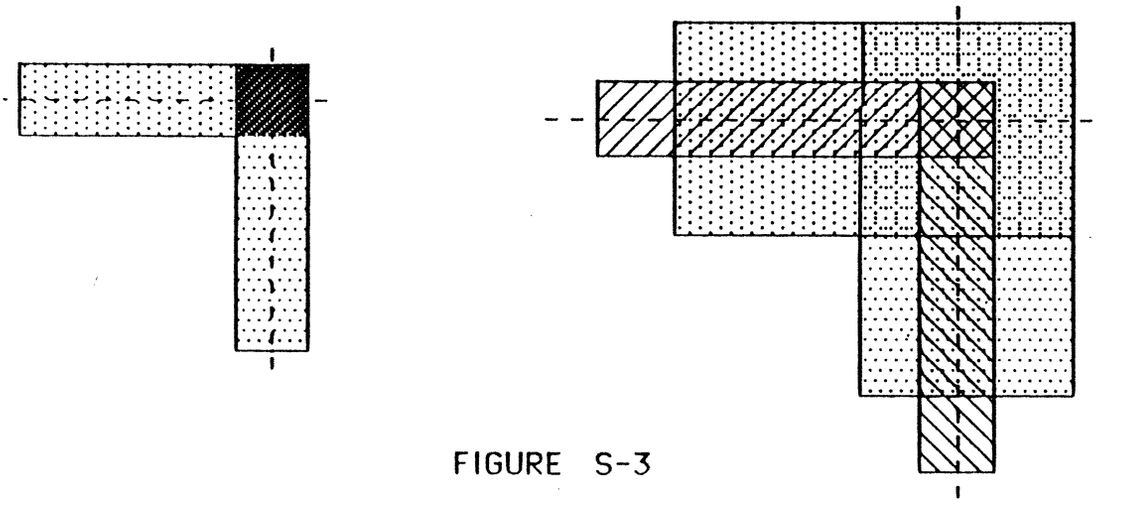


FIGURE S-3

surface estimé à 10-30 % en moyenne.

3.3.2 - Organisation de la cellule

3.3.2.1 - Contexte

Seul le cas des technologies MOS a été traité. Cependant, il semble que ceci puisse aussi s'appliquer à d'autres technologies, dont le bipolaire, moyennant la détection et la symbolisation des fonctions utilisées sur le silicium (178).

D'autre part, pour des raisons techniques, il a été choisi de ne pas utiliser de pointeurs directs sur les objets traités, tous les accès sont effectués via des identificateurs uniques générés par des compteurs.

3.3.2.2 - Objets traités

Les objets traités sont: le segment, la zone et l'appel.
Les segments sont orientés et possèdent des extrémités. Les zones ne sont pas orientées. Il est possible de généraliser complètement, et de ne traiter que le cas de la zone, mais la vitesse d'exécution des algorithmes est alors réduite (FIGURE S-1).

3.3.2.2.1 - Les segments

Les segments sont séparés en deux classes (FIGURE S-2):

-segments simples: constructeur et expanseur simple, un seul niveau technologique.

-segments complexes: constructeur et expanseur complexe,

plusieurs niveaux technologiques (FIGURE S-3).

3.3.2.2.2 - Les zones

Les seules zones traitées sont les zones simples, constituées d'un seul niveau technologique. Les motifs complexes peuvent être réalisés sous forme de cellules particulières que l'on instancie.

3.3.2.2.3 - Les articulations

Les segments, comme les zones, reposent sur des articulations. Ces dernières sont toujours associées à un niveau technologique. Une articulation possède un couple de coordonnées, la liste des symboles qui lui sont afférents.

A une articulation peuvent être posés un certain nombre d'attributs, parmi lesquels l'existance d'un chemin de connexion, la déclaration d'ombres, les expansions, le nommage, le typage, ...

3.3.2.3 - Contenu d'une cellule

Dans la cellule nous avons alors deux champs principaux:

-champ des compteurs:	articulations	C-art
	segments	C-seg
	zones	C-zone
	appels	C-ap
-champ des objets:	articulations	L-art
	segments	L-seg
	zones	L-zone
	appels	L-ap

3.3.2.3.1 - Les compteurs

Les compteurs ont la structure:

- C-* : (valeur . liste-de valeurs-libres)

La valeur est la dernière allouée (entier 16 bits). La liste est l'ensemble des valeurs désallouées lors d'édition. Ceci ne sert que pour un adressage indirect et des statistiques de comptage et n'a de raison d'être qu'au sein d'une maquette. La gestion de la liste des valeurs est extrêmement simple:

- libération d'un objet, mise de sa valeur dans la liste,
- création d'un objet, affectation de la dernière valeur,
- sauvegarde des données, compaction de type garbage collect.

3.3.2.3.2 - Les listes d'objets

Les listes ont la structure:

- L-* : ((m1 . l1) ... (mi . li) ... (mn . ln))

dans laquelle mi est la matière associée aux objets générés et li la liste des éléments générés.

En CMOS, les matières sont, par exemple, pour les segments: -alu2, alu1, poly1, diffp, diffn, ct-poly, ct-diffn, ct-diffp, via, prect, pont, tln, tlp, tcl, tcp, caisson.

Les objets appartenant aux familles diffn, diffp, t*n, t*p portent obligatoirement un attribut "numéro de caisson" pour le CMOS et "implantation ionique" pour le NMOS.

Les articulations sont décrites dans les niveaux technologiques auxquels elles appartiennent (alu2, alu1, poly1, diffn, diffp, caisson, implantation, passivation).

Les niveaux implantation et caisson sont des propriétés d'environnement des objets traités (transistors, diffusions). Dans les objets ils apparaîssent sous forme de référence.
Exemple: un segment transistor "n" porte toujours l'attribut (caisson . xxx).

3.3.2.4 - Structuration de la cellule

On a dans une cellule:

```
CELLULE: -L-ap: ...
         -Corps:((L-art ...) (L-seg ...) (L-zone ...))
         -Compt:((C-art ...) (C-seg ...) (C-zone ...))
```

3.3.3 - Le nommage

Sur tous les objets manipulés, le champ "id" existe ou peut exister. Les objets qu'il contient peuvent avoir un type quelconque, cependant, certains sont pré-définis:

- "string"
- nom indicé : (nom . indice)
- nom préfixé: (préfixe . nom)
- nom matricé: (nom . matrice) ; généralisation des indices

Auquels s'ajoutent: (préfix nom suffix)
(préfix nom matrice)

L'accès aux noms se fait par la fonction unique définie

précédement, laquelle se traduit par l'action:

(**sendq nom** obj-nom objet cellule contexte paramètres)

3.3.4 - Structure des symboles manipulés

Chaque symbole est une suite de séquences de base sur différents niveaux technologiques. De ce fait, il possède les règles standard qui sont liées à ces niveaux.

Il peut présenter des règles d'exception dans certaines configurations (caisson/bodytie). De même il possède les règles électriques (suivi d'équipotentiel ou non) qui permettent les parcours de graphe.

3.3.4.1 - Structure de l'articulation

articulation: (articulation (position ...)
(afferents ...)
(clef ...)
(attributs ...))

"position" est le moyen d'obtenir les coordonnées, en général il s'agit d'un couple (x . y). Il sera vu qu'il peut en être autrement dans le cas d'un symbolique non métrique.

"afferents" est la liste des symboles sur lesquels l'articulation s'appuie.

"attributs" est une A-liste dont les clefs peuvent être:

-expansion: (rectangle ou cercle expansion de l'articulation)

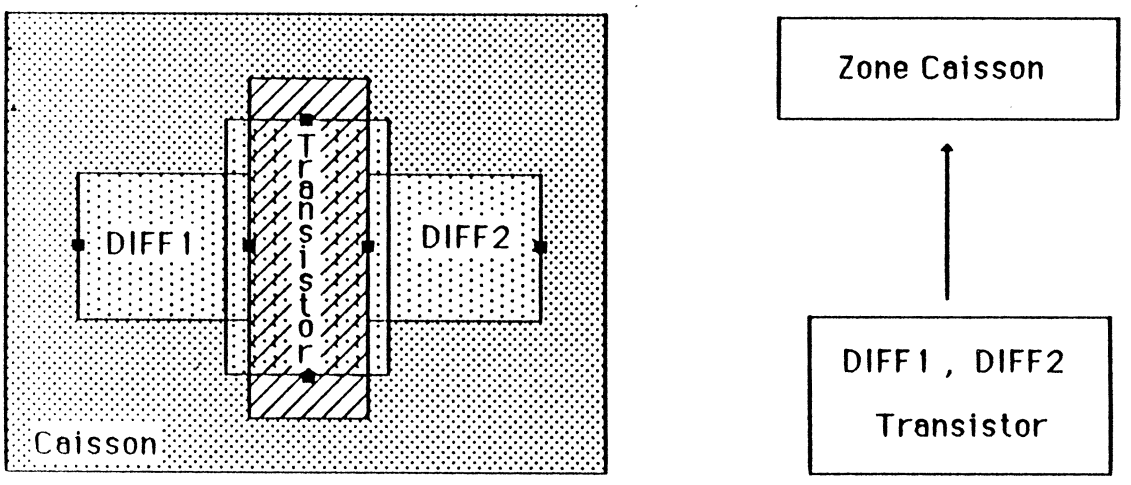
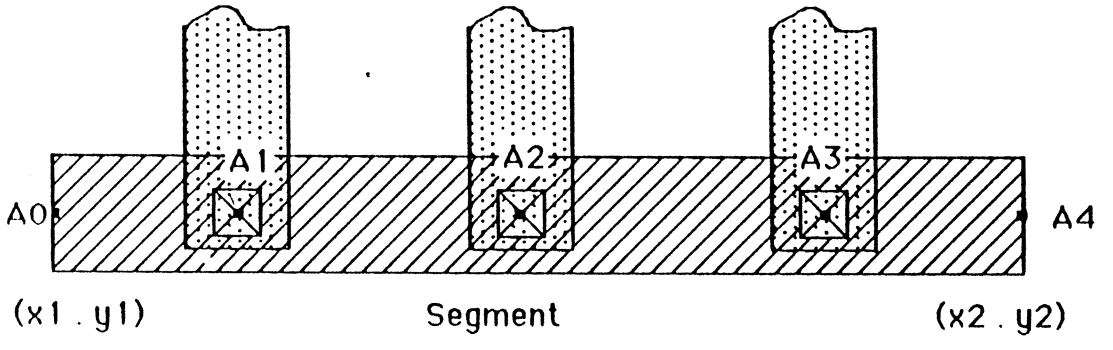


FIGURE S-4




```
(Segment ((x1 . y1) . (x2 . y2))
         (a0 . a4)
         (0)
         (larts (a1 a2 a3)))
```

FIGURE S-5

-id: (cas d'une articulation nommée)

-infos: (paramètres que l'on souhaite accrocher à l'articulation)

-typage: (caractéristique de l'articulation)

-equi: (déclaration d'une appartenance à une équipotentielle)

-chemin: (articulation ayant une signification hiérarchique, et correspondant à un point de connexion d'une cellule appelée).

En conclusion, les articulations sont classées par niveau technologique. Elles peuvent reposer sur un ou plusieurs appels, via les chemins. Elles peuvent être nommées, point de connexion, équipotentielle. Elles portent l'expansion du noeud du graphe qu'elles représentent. Elles connaissent leur environnement. Il est donc possible de savoir si elles sont extrémité de graphe ou simple passage sur une équipotentielle.

3.3.4.2 - Le segment

```
segment: (segment (rect    ...  ) ;rectangle expansion
                  (art1    ...  ) ; extrémité 1
                  (art2    ...  ) ; extrémité 2
                  (angle   ...  ) ; orientation
                  (clef    ...  ) ; propre pointeur
                  (attributs ...))
```

Les attributs sont:

- le caisson: (référence à la zone caisson pour les symboles concernés et situés dans un caisson), (FIGURE S-4)
- Tous les attributs de nommage

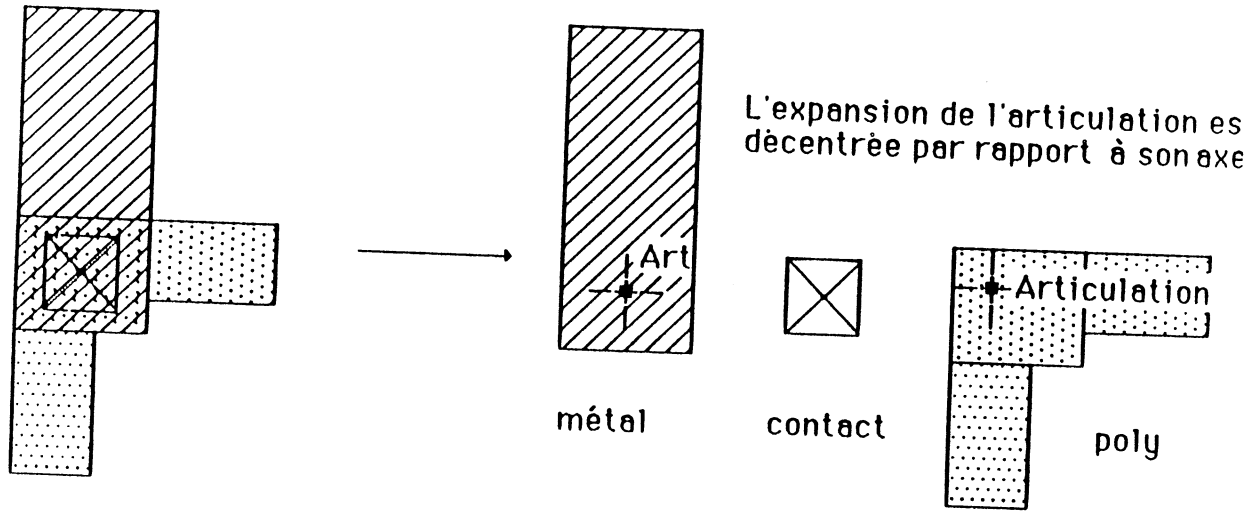


FIGURE S-6

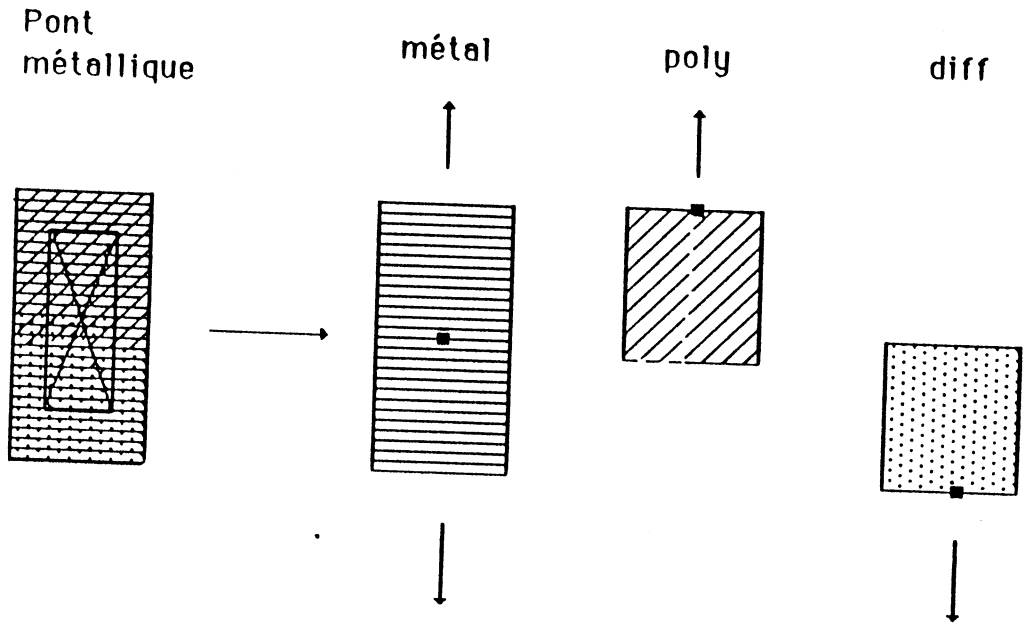


FIGURE S-7

- larts : la liste des articulations situées sur le segment et correspondant à une connexion avec un autre symbole (FIGURE S-5).

Dans le cas de segments complexes, le champ "rect" est remplacé par une A-liste "lrect" constituée des niveaux technologiques et du rectangle expansion.

Il est à noter qu'il est un type de segment particulier, le contact. Le contact est représenté par un objet particulier dans la structure de données, ceci afin de permettre une séparation plus nette entre niveaux technologiques, des parcours d'équipotentielles simplifiés et le dessin de circuits avec positionnement plus fin des contacts. Un contact ne peut être rattaché qu'à deux articulations situées chacune sur un des niveaux mis en jeu (FIGURE S-6).

3.3.4.3 - Structuration de l'appel

Celui-ci est décomposé en deux parties, avec d'un côté tout ce qui est commun aux diverses instances d'une même cellule (cf. chapitre précédant) et de l'autre ce qui dépend de chaque instance: indice, nom, paramètres, transformée géométrique.
On passe d'une instance, à la cellule qu'elle représente, par le couple (chemin d'accès . Cellule).

3.3.4.4 - Symboles complexes

A ces objets, il convient de rajouter les symboles spéciaux, qui dépendent d'une technologie, comme le pont métallique, constitué de diffusion, de poly, de contact et d'aluminium,

soit quatre niveaux différents, d'où quatre rectangles d'expansion. Cependant, les règles technologiques sont telles qu'il n'est accessible qu'une fois par le poly et la diffusion, et indifférement par l'aluminium (FIGURE S-7). D'autres symboles particuliers, ou motifs sont définis en fonction de la technologie. Avec eux sont données la liste des articulations sur lesquelles ils reposent, ainsi que les directions de connexion autorisées (une pour la diffusion et le poly dans le cas du pont métallique).

3.4 - Principe d'édition de la structure

L'édition de la structure se traduit toujours par l'ajout ou le retrait d'un objet. A chaque action de ce type, l'opération est effectuée en deux temps, avec un premier filtrage de tous les objets se trouvant dans une fenêtre dite de pollution autour de l'objet, puis le traitement proprement dit. Ce traitement est fonction des processeurs de vérification utilisés. En général, sont utilisés, l'expanseur de symboles, un vérificateur de règles de dessin incrémental, et un maintien du graphe électrique.

3.4.1 - Suppression d'un objet

Dans le cas de la suppression, le processus est relativement simple et consiste essentiellement à réexpanser les articulations de l'objet retiré, lorsque d'autres objets s'y rattachent. Il convient ensuite de voir si des règles de dessin contextuelles ne sont plus vérifiées après le retrait de cet élément. Dans ce cas, les objets concernés sont mis dans la pile des objets en erreur de garde (cf. Partie 2).

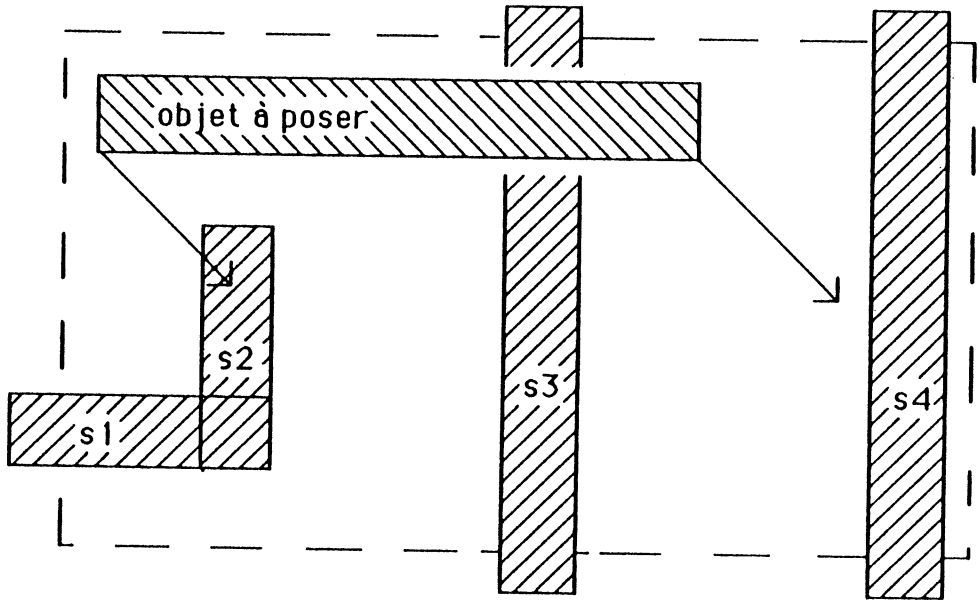


FIGURE S-8

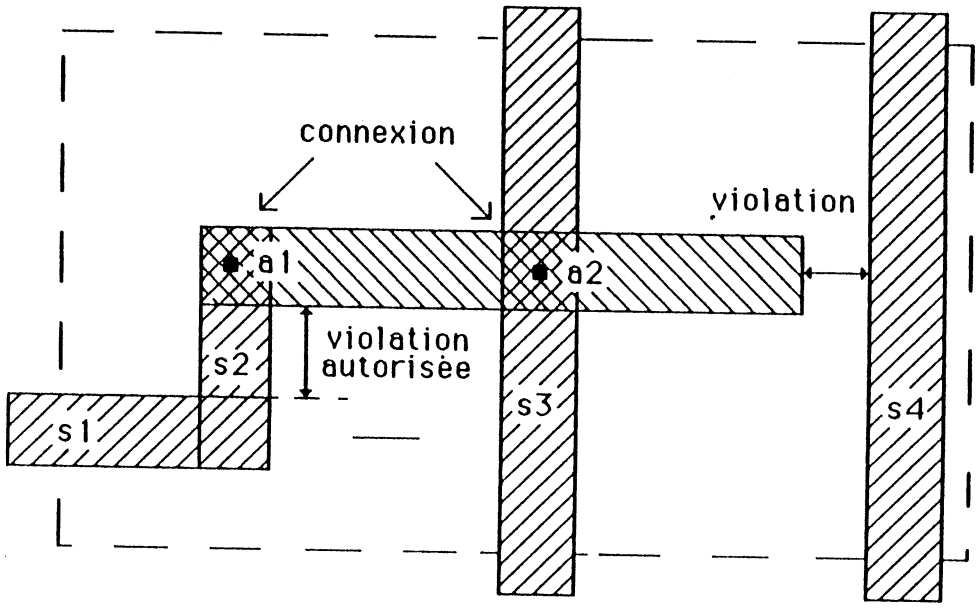


FIGURE S-9

3.4.2 - Ajout d'un objet

Dans le cas d'un ajout d'objet, il convient de rechercher ceux qui présentent une intersection au sens électrique du terme, ceux qui sont à proximité, et ceux qui se rattachent aux mêmes articulations (FIGURE S-8).

Pour ceux qui se rattachent aux même articulations il convient de voir s'ils sont dans une configuration autorisée par le vérificateur de règles (DRC), et de calculer la nouvelle expansion de l'articulation partagée (FIGURE S-9).

Pour ceux qui présentent une intersection, il convient de voir si elle est autorisée. Dans ce cas, il faut générer l'articulation correspondant (FIGURE S-9).

Il convient alors de rechercher les objets se situant en violation de garde des nouvelles expansions des articulations et de l'objet posé. La mise des objets en violation de garde dans la pile des erreurs de garde a lieu, s'ils ne présentent pas une configuration autorisée. La seule configuration autorisée est d'être équipotentiel avec un des objets ayant une articulation commune avec l'objet inséré.

Cette opération est coûteuse. Elle est limitée, de ce fait, au premier ordre, à savoir que seuls les objets partageant une articulation avec ceux au contact direct de l'objet inséré sont tolérés, s'ils sont dans une configuration admise (FIGURE S-9).

3.5 - Langage de génération

La génération de circuits à l'aide d'un langage traduit le désir de "programmer" la création de structures (9). Pour

atteindre ce but, trois approches existent (9):

-l'immersion de primitives de description des objets manipulés dans un langage de programmation: DPL (16), LUBRICK (146).

-la paramétrisation des descriptions par l'ajout de propriétés algorithmiques: L (34), SKILL (190).

-la génération de descriptions statiques par des programmes écrits dans un langage quelconque: LUCIE (128), CIF (156).

3.5.1 - L'immersion

Dans le cas de l'immersion des primitives de description dans un langage algorithmique, ces primitives deviennent des extensions du langage. Cette technique présente l'avantage de n'avoir qu'un langage de programmation pour les différents outils réalisés. L'homogénéité est donc convenablement maîtrisée. De plus, tout l'environnement de création, d'exécution et de vérification est fonction de la puissance du langage, et maintenant les langages de programmation commencent à bénéficier d'un environnement de développement très puissant (SMALLTALK (68), LE_LISP v15.2 (41)) et général avec traceurs, debbug, traitement des erreurs, analyse syntaxique et lexicale, ...

De plus, le descriptif d'un circuit peut être mémorisé sous une forme extrêmement réduite avec l'unique mémorisation du programme de génération.

Les inconvénients de ce choix sont, à priori, la non simplicité des langages de programmation pour les concepteurs. Un interface simplifié est alors à envisager, (ou une formation informatique des concepteurs).

3.5.2 - La paramétrisation

La paramétrisation de descriptions consiste à ajouter des possibilités algorithmiques à un langage de description dans un domaine particulier. Les avantages sont une description concise et la limitation des possibilités algorithmiques aux besoins rencontrés. Les inconvénients sont ceux que l'on rencontre quand on crée un langage algorithmique. Il faut gérer la complexité de traitement. De plus, les mécanismes n'étant pas généraux, mais spécialisés, il finit toujours par apparaître une limitation dans la puissance de paramétrisation. En conclusion, la paramétrisation peut se traduire par l'immersion dans un langage relativement pauvre qu'il convient de gérer. Il faut cependant noter que l'existance de compilateurs de compilateurs peut simplifier grandement la tache.

3.5.3 - Descriptions statiques

La génération de descriptions statiques permet de résoudre les problèmes précités par une simplification extrême du problème. Le langage de description est purement statique, adapté au domaine traité et les contraintes appliquées sont faibles. Le langage de programmation peut être quelconque, la seule contrainte étant la génération de fichiers ASCII de description. Il y a séparation complète entre le langage de description et les langages de génération (Lucie/Lubrick) qui peuvent répondre à des contraintes différentes. Le système d'application est simplifié, car ne traitant que des applications statiques.

Les inconvénients rencontrés sont la génération d'un volume de code important et la maîtrise du code généré.

3.5.4 - Choix

Devant ces trois approches, il apparaît que l'approche par paramétrage doit être abandonnée, l'utilisation d'un sous ensemble d'un langage de programmation pouvant éventuellement pourvoir au problème de la simplification. L'utilisation d'un langage statique ne semble pas intéressante car les langages statiques sont déja nombreux et un standard comme EDIF (61) peut toujours être pris comme langage statique. Les outils correspondant à la génération des circuits existent également. Le langage de programmation va donc être vu dans une approche "immersion lisp". Le langage est traduit par des créations d'objets et par l'envoi de messages à ces objets, ceci afin de bien séparer la manipulation des objets et les constructions algorithmiques. Tous les mécanismes de mise au point du programme et de détection des erreurs sont pris en grande partie en charge par le langage de programmation, tout comme l'analyse lexicale.

3.5.5 - Definition du langage

Comme pour les fonctions de création de cellules par assemblage (chapitre suivant), le langage de génération est vu comme un certain nombre de fonctions lisp manipulant les objets utilisés et les insérant dans le graphe que représente une cellule. Les fonctions sont conçues afin d'être le support textuel de génération et le noyau sur lequel va s'appuyer tout éditeur graphique. Les problèmes rencontrés sont ainsi partagés en un problème d'édition de structure indépendant du mode de génération, et un problème de spécification d'un générateur graphique, les fonctions d'édition existant déja.

Les fonctions de filtrage et de sélection sont les mêmes que celles vues lors de l'assemblage (chapitre suivant), et la syntaxe des fonctions d'ouverture et de fermeture d'une cellule est la même.

La génération d'une cellule correspond à l'exécution d'une procédure lisp. Cette procédure commence par ouvrir la cellule en mode ajout procédural (fonction pose), celle-ci ayant été créée auparavant (fonction donne-cell).

La fonction "pose" insère les objets créés par les procédures de création qui leur sont associées. Celles-ci sont nombreuses, dépendant du nombre de paramètres donné par l'utilisateur, et de l'objet inséré.

Par exemple, sur les segments simples, il est possible de donner comme paramètres:

-le rectangle expansion
-les articulations extrémités
-l'angle du segment

Le programmeur, en envoyant des messages à l'objet qu'il manipule, spécifie le nombre de champs qu'il souhaite donner, puis lance l'opération de pose. Cette opération lance les mécanismes de vérification des gardes et de la cohérence des actions de pose. Tout objet, dont les paramètres donnés ne permettent pas la pose, est mis en pile d'erreur, avec le message associé.

Lors de la fermeture de la cellule, par l'envoi du message de fin d'ajout procédural, les différents flags d'édition

sont testés et la pile d'erreur mise à jour. Les objets en erreur de garde ou de cohérence sont signalés au concepteur.

Les opérations de pose s'appliquent aux segments, aux zones, aux contacts, aux appels et aux symboles définis pour chaque technologie. Les messages envoyés à ces objets sont, tous les messages d'affectation d'une valeur à un de leurs champs.

3.6 - Gestion des erreurs et de la cohérence des circuits

3.6.1 - Gestion des erreurs

Il y a deux types d'erreurs qu'il convient de gérer, les erreurs permanentes liées à l'état de conception d'un circuit au cours des phases de conception (ce type d'erreurs doit être mémorisé, afin de n'avoir pas à les rechercher lors des évolutions du circuit, et afin de pouvoir les retirer lorsqu'elles sont résolues), les erreurs liées à l'exécution d'un processeur donné et à une erreur de construction (ces erreurs ont uniquement un caractère momentané, elles ne sont pas mémorisées lors de la sortie du processeur).

C'est pourquoi les erreurs sont liées aux processeurs. Chaque processeur peut disposer sur une cellule d'un flag, et d'attributs (flags (erreur ...)) et (attributs (erreur ...)). Lors du déclanchement d'une erreur il y a mise du flag à jour avec l'envoi du message erreur avec le nom du processeur et la liste des objets en cause. Ensuite la fonction lisp d'erreur est déclenchée sur l'objet, dans le

contexte traité.

Les erreurs ayant un cararctère permanent sont sauvegardées par la déclaration explicite d'un mode d'impression de ces erreurs. Les erreurs à caractère dynamique n'ont pas de mode d'impression déterminé, et à la sauvegarde elles disparaissent, tout comme les flags de cohérence, ou les données à caractère purement dynamique et qui ne sont pas sauvegardées, mais recalculées si besoin est.

3.6.2 - Maintien de la cohérence

La gestion des incohérences d'un dessin est l'un des points les plus importants d'un système de CAO, car il importe lorsqu'on fait un traitement sur un circuit que le traitement et que les conséquences qui en découlent correspondent effectivement au circuit que l'on a généré. Or, l'expérience montre que pratiquement aucun système n'arrive à gérer cette cohérence. Il y a cohérence lorsque les différentes représentations d'un circuit sont en conformité, et lorsque les niveaux de décomposition hiérarchique d'un circuit sont en conformité avec le niveau inférieur correspondant.

La gestion de la cohérence d'un circuit se décompose en deux parties: la détection de l'incohérence, et la résolution de cette incohérence.

3.6.2.1 - Détection

Dans le cas du dessin des masques, la détection est extrêmement simple, et se fait par un simple parcours de

l'arbre des appels avec vérification des dates.

A chaque définition de cellule est associée, pour les processeurs de génération de circuits, une valeur "version". Cette valeur est la date d'édition ou de génération de la cellule.
De la même façon, les instances de cellules possèdent toutes un champ "version".
En comparant les deux valeurs de ces champs, il est possible de savoir si une instance est antérieure ou postérieure à la génération ou l'édition de la cellule. Si l'instance est postérieure, la hiérarchie est cohérente. Si elle est antérieure, elle est incohérente.

La détection des incohérences de "layout" se fait par un parcours hiérarchique récursif de l'arbre des appels. Ce traitement s'applique particulièrement bien sur la structure définie, et demeure extrêmement performant.
Lorsqu'une cellule présente une incohérence, il y a déclanchement d'erreur avec pour information: la cellule concernée et le processeur de recherche de cohérence.

3.6.2.2 - Résolution

Le rattrapage d'erreur lié au processeur de cohérence a été vu sous un angle de simplicité, car le retour à une cohérence dans un circuit n'est pas forcément voulu ou intéressant pour le concepteur. La résolution d'une incohérence momentanée et de bas niveau peut se transformer rapidement en la transformation de l'ensemble de la description du circuit si le concepteur n'y prend garde. C'est pourquoi l'algorithme de cohérence ne peut être

systématique.

Lorsqu'il est déclanché, le type d'action effectué dépend de la cellule où une incohérence est détectée.

3.6.2.2.1 - Cas de la génération par procédure

Lorsque celle-ci a été générée par assemblage, ou par procédure, soit de manière reproductible; comme dans le mécanisme "MAKE" utilisé à la compilation dans le système UNIX (171), il y a réexécution de la procédure de génération et écrasement de l'ancienne structure. Ceci sous entend qu'à partir de la structure de données qui représente un circuit, il est possible de retrouver le programme générateur. Cette opération peut être réalisée de façon manuelle, en demandant au concepteur de regénérer la cellule mise en cause, ou de façon automatique en mémorisant le fait qu'une cellule est générée automatiquement, et le lieu physique du générateur. L'approche qui consiste à conserver les constructeurs dans la structure ne semble pas satisfaisante, car bien souvent ce sont les constructeurs qui ont été modifiés lors de la modification de structure qui entraîne une incohérence.

3.6.2.2.2 - Cellule générée manuellement

Lorsque la cellule est une cellule qui a été éditée, l'instance de la cellule mise en cause est retirée, puis reposée, ceci afin de recalculer la nouvelle vue externe (chapitre suivant) de la cellule. Il est alors possible de voir si la vue externe de la cellule éditée a changé, en comparant les deux vues, avant et après repose. Si oui, alors il y a répercussion hiérarchique montante de

l'incohérence, si non l'incohérence est résolue.

3.7 - Vérification hiérarchique des règles de dessin

Dans tous les systèmes de CAO, la vérification hiérarchique des règles de dessin (DRC) est incontestablement le point faible, entraînant soit des restrictions draconniennes quand aux possibilités de dessin, soit des erreurs de garde non résolues et détectées, ou encore la détection d'erreurs qui n'existe pas. D'autre part, un vérificateur de règles de dessin "on-line" devient coûteux pour de grosses hiérarchies de dessin. Il convient donc d'utiliser ce dernier pour de petites hiérarchies, ou pour un niveau de hiérarchie. Il devient alors nécessaire de posséder un outil "off-line" qui permet de vérifier la justesse du dessin, et qui soit moins coûteux que les vérificateurs classiques.

3.7.1 - Pollution d'une cellule

Si un système hiérarchique représentant les circuits sous forme d'un arbre d'appels est utilisé, il est possible de décomposer les problèmes de règles de dessin en problèmes de pollution de l'espace d'une cellule (rectangle enveloppe - Bbox).

Est appelée "pollution" tous les objets d'une cellule qui viennent en interaction avec le corps d'une autre cellule. Il s'agit donc d'une liste d'objets.

La pollution propre d'une cellule se décompose alors en:

- -la pollution "père/fils".
- -la pollution due aux frères et descendants des frères qui ont déja été traités.

-l'héritage de la pollution propre du père; Aspect récursif, car le père est traité avant le fils.

Lorsqu'on traite un objet, on cherche à résoudre sa pollution propre, tout en se préparant à descendre dans l'arbre d'appels. d'ou les actions:

1-Traitement de la pollution propre d'une cellule.

2-Création de la pollution de la cellule sur les cellules soeurs et sur les soeurs de ses ancêtres non encore traités.

3-Descente dans l'arbre d'appel ou passage à la cellule soeur suivante si descente impossible.

La pollution, intersection des encombrements de cellules peut être représentée par un simple pointeur sur la cellule polluante et la transformée géométrique associée aux placement relatif des deux cellules.

3.7.2 - Vérification des règles sur les objets polluants

La vérification se fait en déterminant l'ensemble des objets polluant l'espace de la cellule, en leur appliquant la transformée géométrique correspondant au placement relatif des deux cellules, et en vérifiant le graphe unique qui en résulte dans l'espace de la cellule. On est revenu au cas de la vérification dans une cellule.

Ce processus de vérification des règles de dessin est performant par rapport à un processus classique avec mise à plat de tout le circuit, cependant il peut être relativement amélioré par une méthodologie de dessin et l'utilisation d'outils annexes.

3.7.3 - Accélération du processus de "DRC"

Tout d'abord, la pollution "père/fils" n'est vérifiée qu'une fois. Il y a marquage sur l'arbre d'appels, et lorsque il y a passage sur une cellule déja traitée, seul le nouvel environnement de la cellule est traité.

3.7.3.1 - Utilisation des ombres

Cette même pollution "père/fils" peut être en grande partie simplifiée par l'existance de "prolongés" (chapitre suivant). Ces objets permettent de savoir comment on va venir se connecter à une cellule. Lorsque un élément de pollution correspond à un prolongé, il est considéré comme valide. Il est retiré de la liste des objets polluant la cellule.

3.7.3.2 - Utilisation des frontières

D'autre part, l'existance de frontières à une cellule (chapitre suivant) permet de déterminer de manière plus précise les éléments qui sont réellement en violation avec l'espace de la cellule. Le traitement est donc réalisé en deux phases; détermination des cellules ayant une intersection au niveau des Bbox, parmi ces cellules, détermination des cellules ayant une intersection au niveau des frontières, (ce sont les seules cellules ayant une pollution possible).

Le dernier moyen d'accélérer le processeur de vérification des règles de dessin hiérarchique est la création de contextes d'appel lors de la création des cellules (cf. ombres). Lors de l'instanciation d'une cellule, il suffit de

vérifier que les objets de pollution ont été déclarés sous forme de contexte. Si c'est le cas, la pollution est autorisée.

4 - Symbolique non métrique

Le symbolique, tel qu'il a été défini, présente l'intérêt de permettre la génération de circuits d'une très grande densité, mais présente tous les inconvénients d'une représentation purement métrique. Il semble donc intéressant de voir comment sur la même structure de données, et à partir d'une simple extension du langage, il est possible de définir un symbolique non métrique.

4.1 - Utilisation

Par opposition au symbolique "dur", le symbolique "mou" n'a pas en lui de structure métrique figée. Il doit être, comme son nom l'indique, facilement déformable. Sa vocation est multiple.

4.1.1 - Démarche descendante

Le but est de permettre une démarche de conception "top-down" dans laquelle on est amené à utiliser et manipuler l'enveloppe d'une cellule avant même de connaître son contenu avec précision, hormis sa fonctionnalité et ses performances. On est amené à déformer cette cellule afin de l'insérer dans un contexte global en minimisant les pertes de surface liées aux interconnexions. La résolution d'une cellule "molle" asociée à un certain nombre de paramètres le permet plus facilement.

4.1.2 - Génération automatique

Le symbolique "mou" a pour objectif de devenir le support de générateurs de "layout", en précisant le type de cellule que l'on souhaite réaliser et en donnant une idée approximative du positionnement topologique des divers éléments constituant la cellule. Les générateurs deviennent alors des algorithmes simples (compactage, assemblage) qui n'ont pas à prendre de décision "intelligente", les positionnements sont prédéfinis, ils n'ont qu'à résoudre les contraintes topologiques (connexions, largeur, hauteur, performances) imposées à la cellule molle.

4.1.3 - Intégration des outils

Le dernier but, du symbolique "mou", est de devenir la structure de données sur laquelle un certain nombre de processeurs, n'ayant pas besoin d'une information topologique exacte peuvent s'appliquer (simulateurs switch, réseau électrique, ...)

4.2 - Spécification

On peut définir un symbolique mou par le formalisme de relations de proximité entre les divers éléments manipulés. Ainsi, dans une représentation "manhattan", il est possible de déterminer les relations de gauche, droite, haut, bas (158) entre symboles, ou bien d'autres représentations, comme la tuile de Magic (125), peuvent être utilisées. Le symbole lui-même doit être typé, soit vertical, soit horizontal, lorsqu'il est possible de lui donner une direction privilégiée (exemple: le segment d'aluminium).

A ces notions purement topologiques, doivent s'ajouter des notions ayant également une signification électrique (la taille des transistors, la largeur des fils, ...), des notions de contraintes de positionnement entre éléments d'une représentation, et des contraintes de performance déterminées par l'existance de relations entre la taille des différents symboles, et les performances d'une cellule.

La génération d'une cellule molle devient alors un simple positionnement relatif entre symboles (sans préoccupation aucune quand aux règles technologiques et au placement fin des symboles assemblés) et la détermination des contraintes topologiques et de performances. Le passage de la cellule "molle" à un "layout" se fait par la résolution de ces contraintes.

4.3 - Intérêt pour une génération semi-automatique

Un tel symbolique "mou" possède le grand intérêt d'accélérer fortement la conception d'un circuit. En effet, à partir d'un graphe électrique ou d'une représentation logique, il est assez facile de générer automatiquement une première version de cellule molle. Un placement approximatif des objets manipulés est toujours possible pour résoudre le graphe des contraintes. Ce placement correspond souvent à la plus grande partie du temps de conception lors d'une action purement manuelle. Il suffit, ensuite, de retoucher le résultat obtenu, de déclarer les contraintes que l'on souhaite voir imposer à la cellule et de les résoudre pour obtenir le circuit voulu. Le passage d'un niveau de description supérieur au niveau du "layout" se traduit alors par l'ajout d'informations à une structure de données déja

connue, et non par la répétition d'informations comme c'est bien souvent le cas. D'autre part, un prépositionnemnt automatique des différents éléments d'un dessin, accélère grandement la vitesse de dessin, comme les expériences le démontrent.

4.4 - Rapports avec le symbolique métrique

4.4.1 - Utilisation de la structure définie pour le métrique

A partir des spécifications données au 4.2, il apparaît que la simple affectation des champs direction, rectangle et larts dans la structure de données du symbolique métrique le transforme en symbolique mou. Les contraintes peuvent alors être insérées dans le graphe sous forme de relations établies entre les articulations. Pour cela il suffit de typer le champ coordonnées des articulations, et d'y mettre l'information que l'on souhaite (position relative de deux articulations, distances min ou max entre deux points).

4.4.2 Passage d'une représentation "molle" à une "métrique"

La résolution du graphe correspondant au symbolique "mou" se fait en tenant compte des contraintes, taille des objets (w/l ou largeur de fil) et des relations traduites par le graphe du circuit. L'utilisation d'un compacteur, et l'affectation des valeurs de contraintes permet la génération d'un circuit.

L'intervention manuelle pour orienter la compaction peut également se faire en introduisant des objets qui vont forcer certaines compactions (introduction d'un coude,

brisure d'un fil).

4.4.3 - Intérêt d'une struture commune

Un système symbolique où la représentation métrique et la représentation "molle" reposent sur la même structure, et pratiquement le même langage (seules les coordonnées ne sont pas fournies, les articulations ne sont que des objets partagés ayant des positions relatives) présente l'avantage important de permettre à de nombreux outils de fonctionner à tout niveau de spécification du "layout". De plus, les contrôles successifs sur le circuit, après évolution, peuvent être facilement effectués, par des méthodes différentielles. La structure de donnée définie est adaptée à la réalisation d'un simulateur SPICE (121) ou d'un simulateur hiérarchique tel que le définit J.P.Caisso (37). Seul, un processeur de simplification de la structure, afin de regrouper les noeuds électriques, et de classer les éléments actifs, peut être ajouté, ce qui accélère considérablement les processeurs de parcours d'équipotentielle.

5 - Vision externe d'une cellule

5.1 - Introduction

En reprenant les caractéristiques d'un circuit intégré, telles que les ont définies J.P.Schoellkopf et F.Anceau pour les logiciels LUCIE (128) et LUBRICK (146), ou J.A.Rowson (142), il est possible de dire qu'un circuit est une hiérarchie de figures, dans laquelle les feuilles de

l'arborescence sont de petites figures, appelées cellules, qui peuvent être dessinées à la main (symbolique métrique), conçues de façon optimale et pour s'assembler entre elles par aboutement. Il est donc possible de décrire la hiérarchie qui représente un circuit, ou des blocs de cellules, sous forme d'un programme écrit dans un langage de haut niveau. Cette approche est de plus en plus courante, et on la trouve dans de nombreux systèmes. Nous ne citerons que LUCIFER de l'INRIA (43), ou LUBRICK de l'IMAG(146).

Un seul élément est manipulé. Il s'agit de la cellule. Cette dernière est métrique et caractérisée par certains attributs qui constituent sa vision externe.

5.2 - Vision externe d'une cellule

Une cellule est vue sous la forme d'un graphe électrique, auquel on peut accéder par un ensemble de points, les points de connexion. D'autre part, une cellule correspondant à un descriptif topologique, elle occupe dans le plan une certaine surface, qui est traduite par une Bbox, et pour chaque niveau technologique, par une frontière. Enfin, l'accès aux points de connexion d'une cellule, ou le passage à travers une cellule, présentant un problème de violation de l'espace de la cellule, un ensemble de chemins d'accès est décrit sous forme de symboles particuliers (ombres de connexion ou de passage). A ceci, il convient d'ajouter la possibilité de pouvoir attribuer à une cellule toutes les informations que le concepteur, ou les logiciels utilisés, souhaitent. Ceci peut être fait sous forme d'attributs.

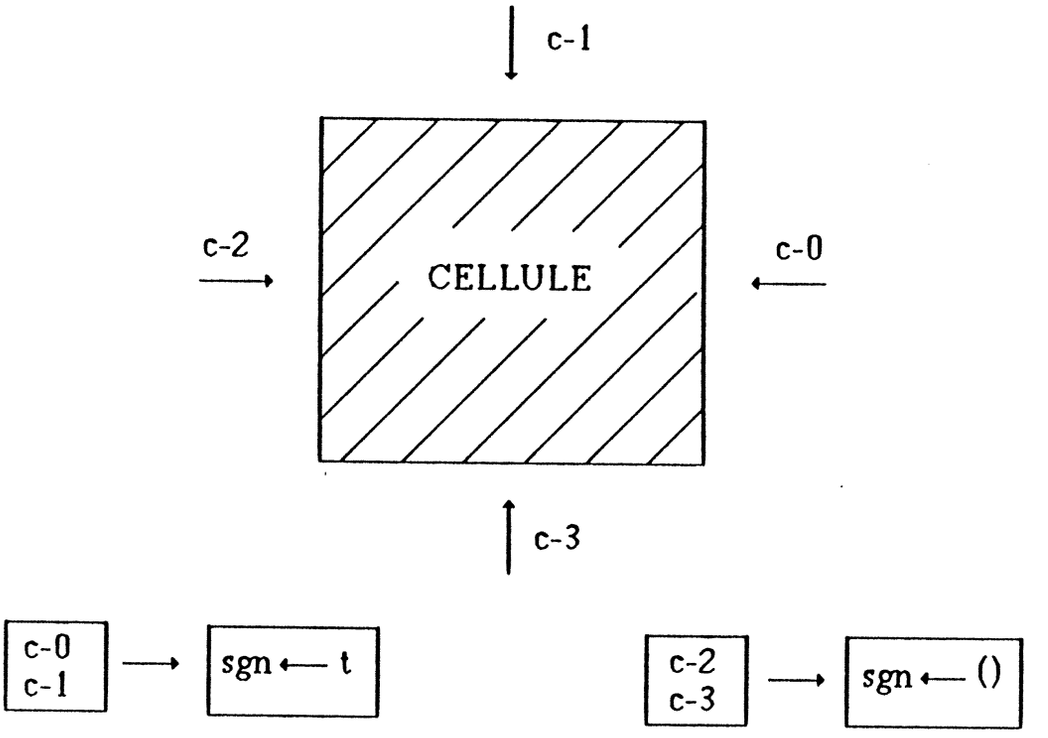


FIGURE F-0

Sur un niveau (mat-i) donné

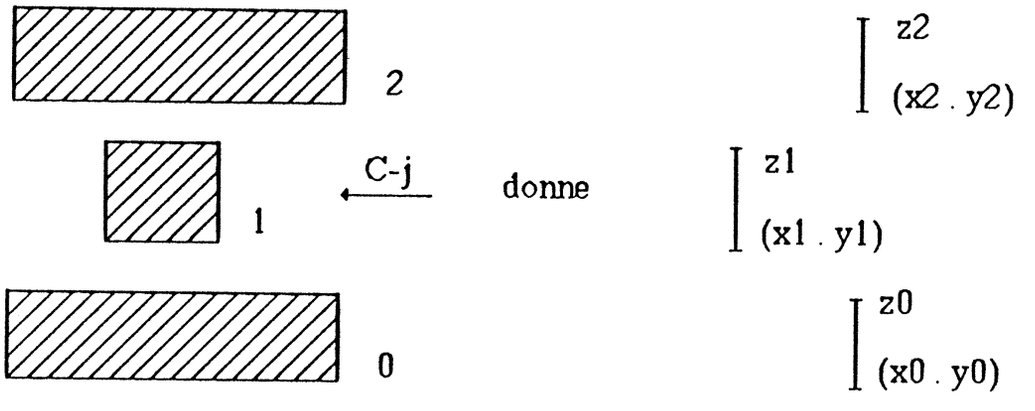


FIGURE F-1

5.3 - Frontières d'une cellule

Les frontières sont conçues pour avoir un rôle multiple dans le système. Tout d'abord un rôle de validation hiérarchique incrémentale lors de l'insertion d'appels de cellules dans le cadre d'une édition, ensuite le rôle de mise à la garde lors du positionnement de cellules (assemblage, compactage). Enfin, la simplification des algorithmes de vérification des règles de dessin hiérarchique. Elles permettent le contrôle du respect des règles technologiques de manière hiérarchique et la possibilité de concevoir des circuits avec recouvrement partiel de cellules. De leur spécification et de leur implantation dépend la puissance et l'efficacité d'un certain nombre d'outils (DRC, assembleur, ...).

5.3.1 - Spécification

Les frontières sont l'extraction du layout vu des quatre côtés d'une cellule, dans une représentation manhattan (est: c-0, nord: c-1, ouest: c-2, sud: c-3) (FIGURE F-0). Elles peuvent être plus ou moins lissées suivant le contexte (technologie, nombre d'élements, algorithme de lissage choisi, ...).

Les frontières d'une cellule sont spécifiées par une liste de frontières par niveau technologique (FIGURE F-2):

Chaque frontière "fi" a quatre bords constitués d'arêtes (FIGURES F-1 et F-3), une arête étant un couple ((x . y) . z), z étant du type coordonnée (x ou y). Ce codage a été choisi, car il est le plus performant dans le cadre des algorithmes réalisés.

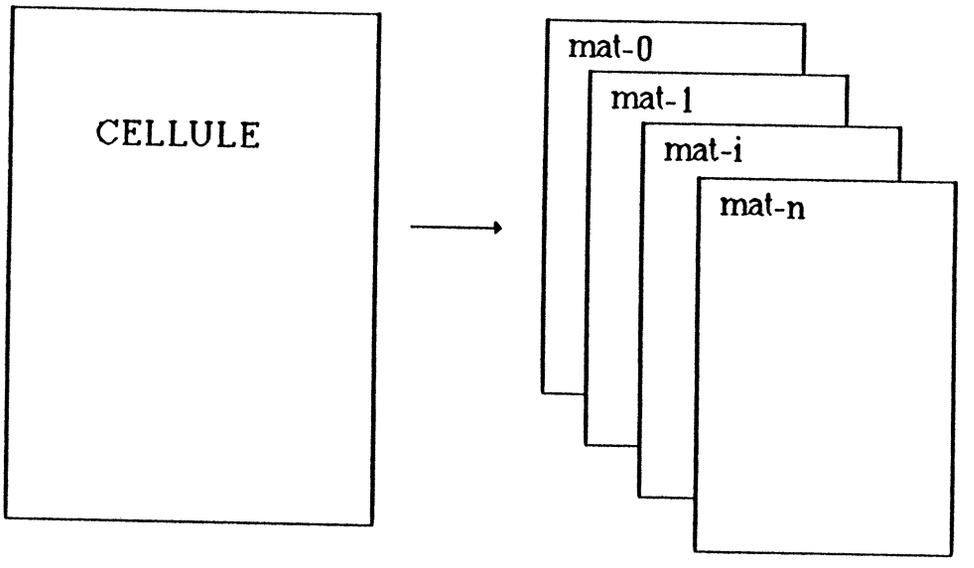


FIGURE F-2

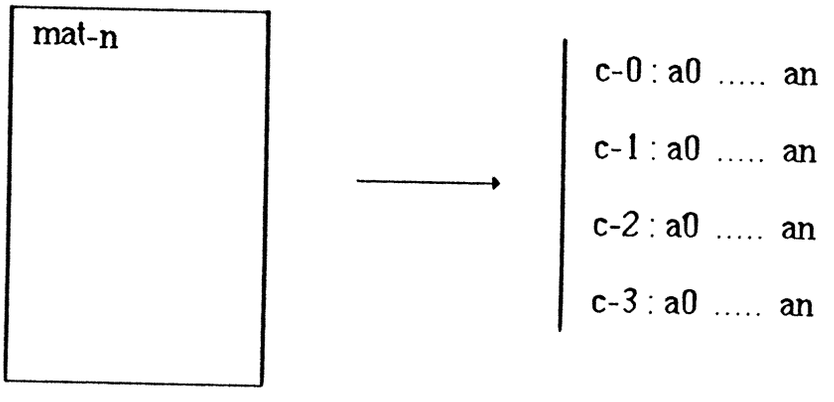


FIGURE F-3

La structure suivante est obtenue:

- FRONTIERE : ((mat-1 fr) ... (mat-i fr) ... (mat-n fr))
- fr : ((n . l-a) (s . l-a) (e . l-a) (o . l-a))
- l-a : (arête-1 ... arête-i ... arête-n)
- arêtei : ((x . y) . z)
- z : x / y

Les frontières sont considérées comme tous les autres symboles manipulés, de ce fait elles doivent respecter des règles technologiques identiques à celles respectées par les autres symboles (gardes entre niveaux de même matière, et niveaux inter-agissants).

5.3.2 - Fonctions sur les frontières

Il y a quatre types de fonctions sur les frontières, les fonctions: de CREATION, de LISSAGE, d'UNION, et de MISE à la GARDE.

5.3.2.1 - Union de deux frontières f1 et f2

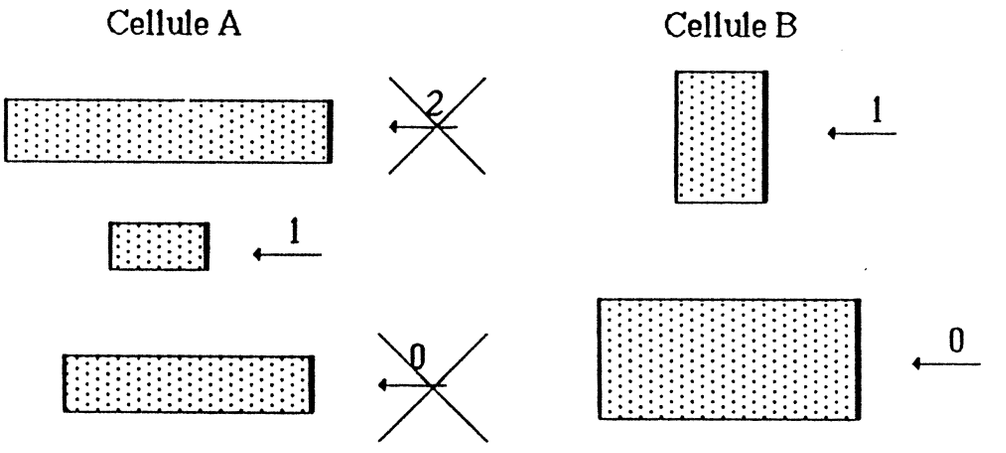
L'union F de deux frontières f1 et f2 se traduit par une union, niveau technologique par niveau technologique, de f1 et f2.

Quelle que soit la matière "mat-i", la frontière associée à F est:

F(mat-i)=UNION(f1(mat-i),f2(mat-i))

Avec: F(mat-i):((n . la)(s . la)(e . la)(o . la)), la liste des arêtes.

Union des Frontières (vue du coté c-0)



La Frontière Union de A et B a pour arêtes sur c-0 :
c-0(AuB) = (A(1), B(0), B(1))

FIGURE F-4

Ajout d'arêtes à une liste, position relative au premier élement de la liste

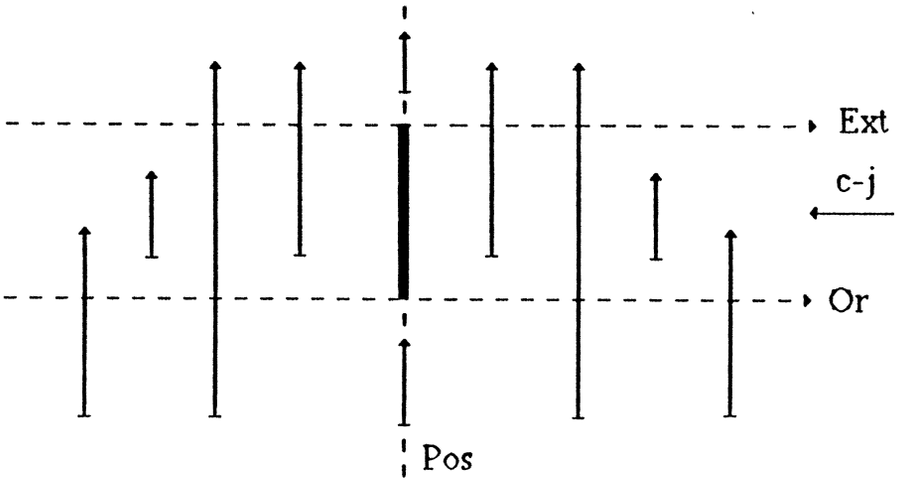


FIGURE F-5

Ce qui se traduit, sur chaque côté c-j (0, 1, 2, 3) par:

la(F,mat-i,c-j)=UNION(la(f1,mat-i,c-j),la(f2,mat-i,c-j)

Quelle que soit l'arête de f1 ou f2, si elle est visible dans la direction c-j, alors elle devient arête union (FIGURE F -4).

5.3.2.2 - Fonction de création

Cette fonction a été implantée sous la forme d'une fonction d'ajout d'un élement à la frontière, les conditions initiales pouvant être une frontière vide. Cette méthode n'est pas la plus performante. En revanche, elle est très simple d'implantation, demeure générale et nécessite un volume de code réduit.

Plusieurs cas ont été envisagés pour le codage des arêtes d'un côté "c-j" d'une frontière (arêtes recouvrantes, classées par origine et extremités croissantes, par position topologique, pour minimisation du nombre d'éléments traités, ...). Il est apparu que lorsque le nombre d'arêtes utilisé pour représenter une configuration donnée diminue, la complexité des algorithmes croît de façon considérable et le temps d'exécution devient catastrophique, d'où l'établissement d'une loi de simplification des algorithmes. Toute arête visible est conservée, celle-ci ne pouvant être visible par discontinuité (FIGURE-6). Pour des raisons de simplicité, la solution choisie est la suivante:

-les frontières ne sont pas des suites continues d'arêtes, mais les arêtes ne peuvent se chevaucher,

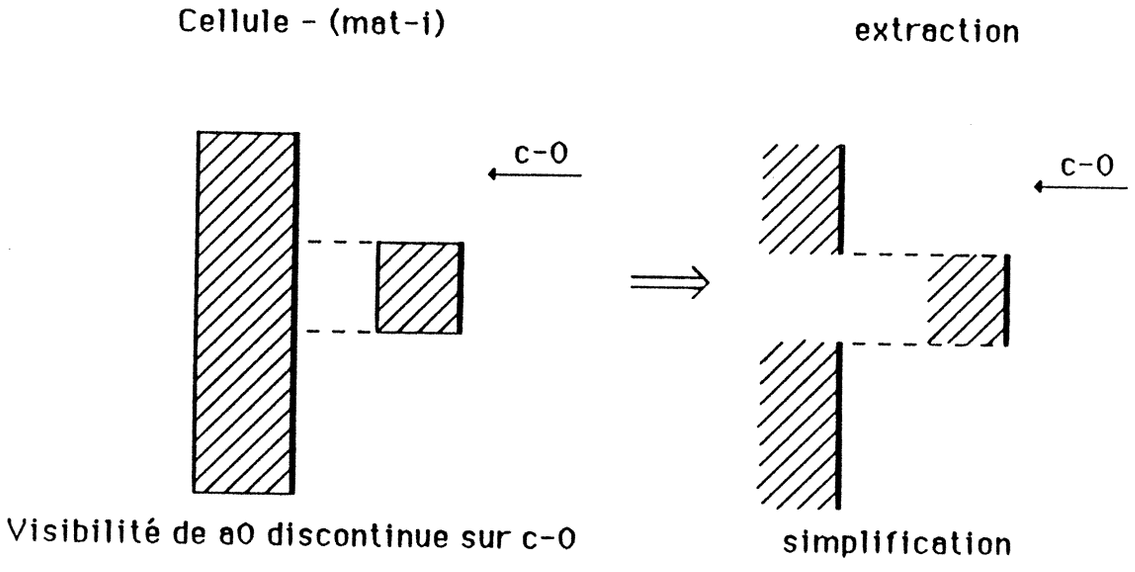


FIGURE F-6

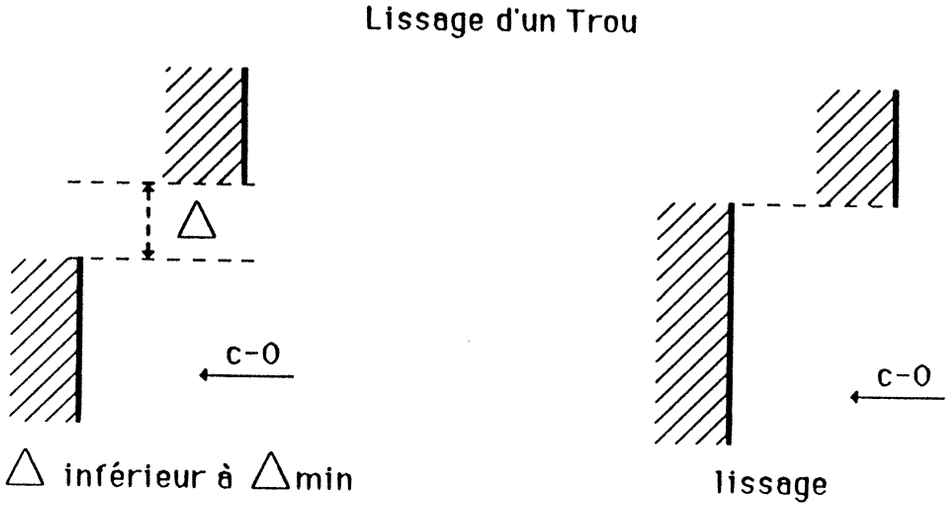


FIGURE F-7

soit: l'origine d'une arête est toujours supérieure à l'extrémité de l'arête qui la précède.

-la longueur d'une arête a une taille minimale déterminée soit par la technologie, soit une valeur minimale fournie par l'utilisateur.Il en est de même pour les trous.

-un trou signifie la non présence de matière sur une bande correspondante, une transparence rectiligne.

Finalement, l'opération d'ajout d'un élément à une frontière se traduit par l'opération unique d'ajout d'une arête à une liste d'arêtes (FIGURE F-5), et ceci sur les quatre côtés. La réalisation peut se faire sous forme d'un algorithme simple général, et relativement performant:

```
(de :ajoute (liste or ext pos sgn)
  (cond ((gt or (:ext (car liste)))
         (if (cdr liste)
             (:ajoute (cdr liste) or ext pos sgn)
             (:insere-1 liste or ext pos)))
        ((lt ext (:or (car liste)))
         (:insere-2 liste or ext pos))
        ((gt or (:or (car liste)))
         (cond ((funcall (:ft sgn)(:pos (car liste)) pos))
                (when (lt (:ext (car liste)) ext)
                  (if (cdr liste)
                    (:ajoute (cdr liste) or ext pos sgn)
                    (:insere-3 liste or ext pos))))
               ((funcall (:ft sgn)(:pos (car liste)) pos)
                (cond ((gt (:ext (car liste)) ext)
```

```
                    (:insere-4 liste or ext pos sgn))
                   (t (:insere-5 liste or ext pos))))
            (t (:valide liste or ext pos)))
      ((ge ext (:ext (car liste)))
       (cond ((funcall (:ft sgn) (:pos (car liste)) pos)
              (when (gt or (:or (car liste)))
                    (:insere-6 liste or ext pos))
              (when (gt ext (:ext (car liste)))
                (if (cdr liste)
                  (:ajoute (cdr liste) or ext pos sgn)
                  (:insere-7 liste or ext pos))))
             (t (ochangeq arete (car liste)
                  or or ext ext pos pos)
                (:valide liste or ext pos))))
      (t (if (= pos (:pos (car liste)))
           (:or (car liste) or)
           (:insere-7 liste or ext pos sgn)))))
```

Dans lequel "liste" est la liste des arêtes taitées, "or" "ext" et "pos" sont le descriptif de l'arête que l'on souhaite insérer, et "sgn" est soit la valeur "true", soit la valeur "nil", suivant que la liste traitée correspond aux côtés c-0 et c-1, ou c-2 et c-3.

Les fonctions (:insere-* liste or ext pos) sont de simples fonctions d'ajout d'un élément à une liste. La fonction (:valide liste or ext pos) est une fonction d'ajout à une liste et de vérification de cohérence de la liste.

5.3.2.3 - Lissage d'une frontière

Le lissage d'une frontière se fait à plusieurs niveaux et

Lissage de frontières

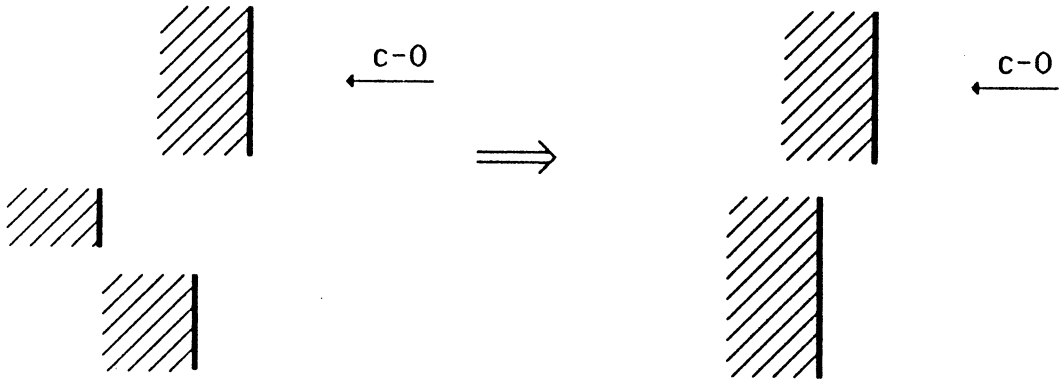
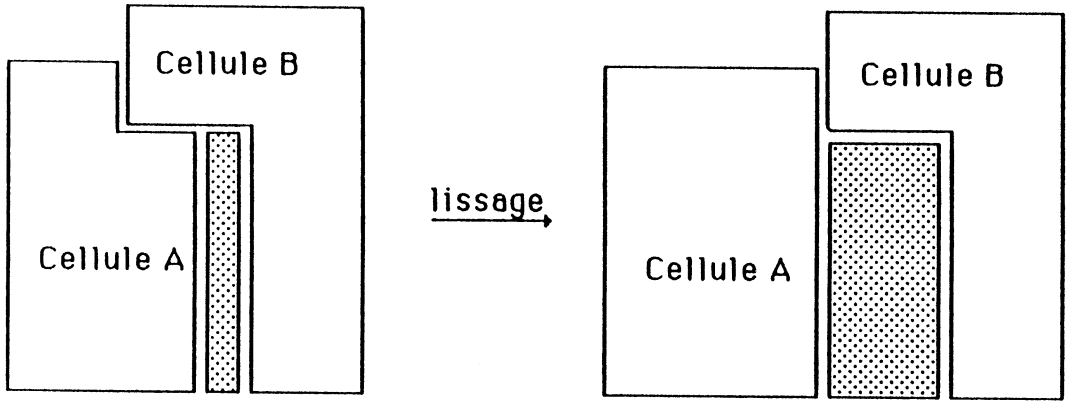


FIGURE F-8

Assemblage de deux cellules A et B



assemblage sans lissage

assemblage avec lissage de la cellule A

FIGURE F-9

peut avoir des objectifs différents. Le premier lissage qui existe, est géré obligatoirement par le système, les autres le sont avec une possibilité d'intervention de la part de l'utilisateur.

5.3.2.3.1 - Premier lissage

Son but est de faire d'une frontière un ensemble cohérent de données. De ce fait, il est utilisé pour éliminer toutes les configurations de frontières n'ayant aucune incidence sur la vision externe d'une cellule. Ainsi, lorsqu'un trou apparaît dans une frontière, et qu'il est d'une largeur inférieure à la garde admise autour d'un élément de même matière pour une technologie donnée, il convient de le faire disparaitre, afin de simplifier les divers algorithmes d'assemblage et de DRC hiérarchique (FIGURE F-7). De même, lorsqu'une arête présente une largeur insuffisante, elle est éliminée, au profit des frontières voisines (FIGURE F-8). Il n'y a pas d'algorithme de lissage à proprement parler, car ce premier algorithme est inclu dans l'algorithme de création pour des raisons d'efficacité.

5.3.2.3.2 - Autres lissages

Leur but est toujours de simplifier la représentation externe d'une cellule, seulement leur exécution signifie que lors de la création d'un circuit, des erreurs de garde peuvent être signalées de façon érronée. Des erreurs de placement importantes peuvent être faites lors de l'assemblage de petites cellules, avec une perte de place non négligeable (FIGURE F-9). C'est pourquoi il semble nécessaire de fournir à l'utilisateur la possibilité

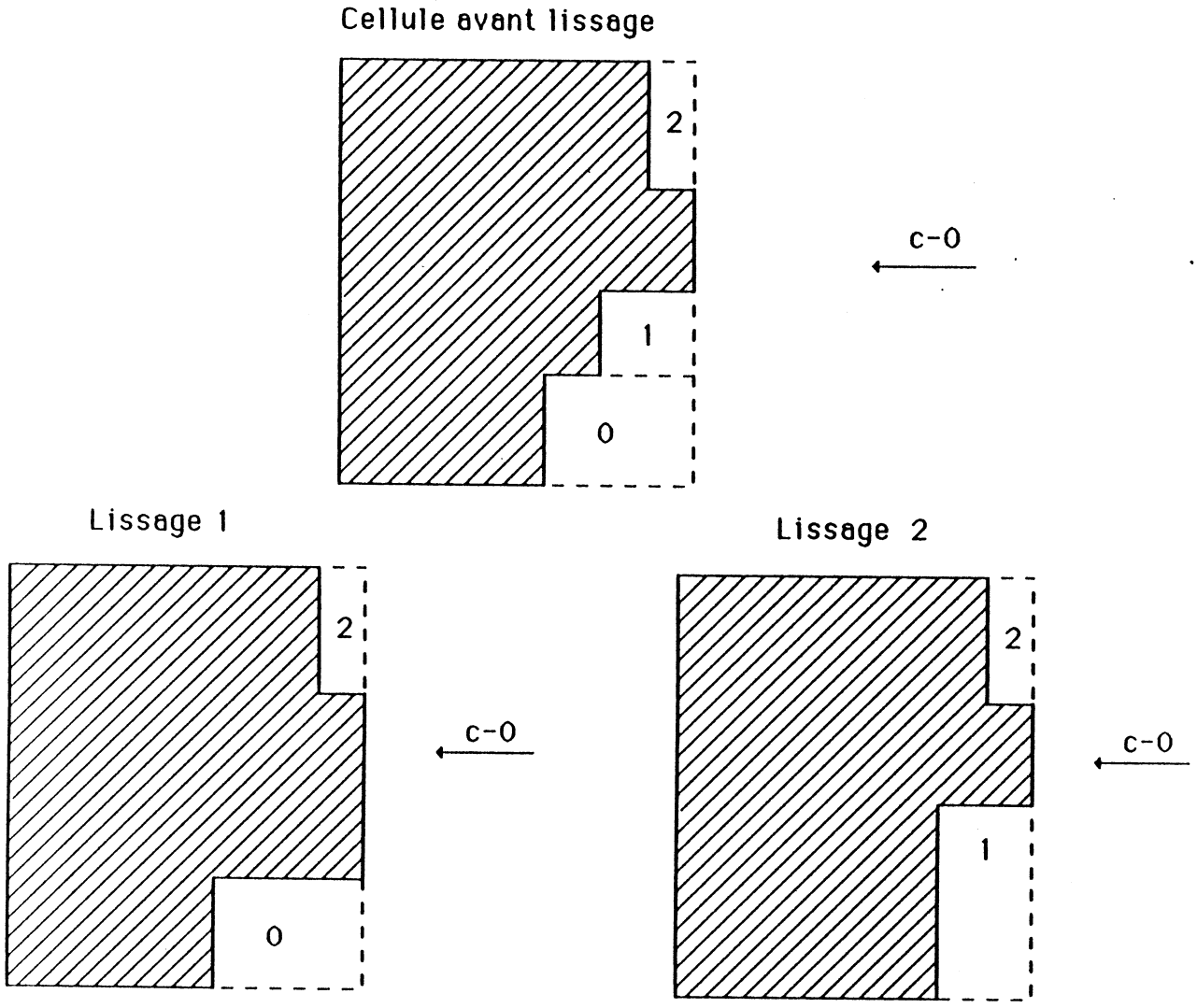


FIGURE F-10

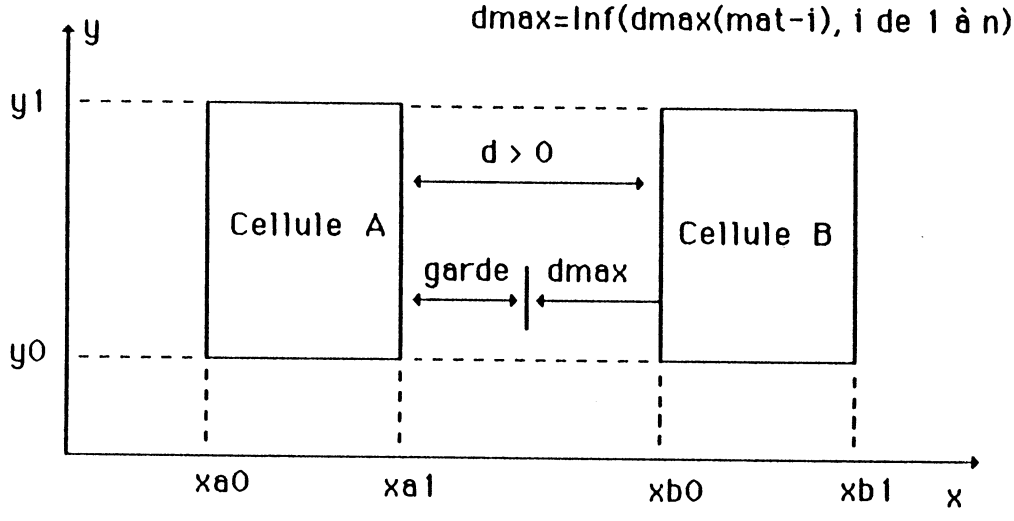


FIGURE F-11

d'intervenir à ce niveau, par le choix des algorithmes de filtrage, ou par le choix des paramètres associés. Ainsi, par exemple, il est possible de refaire le filtrage de base en changeant la valeur "min" admise, et ceci de façon dynamique. D'autre part, l'algorithme de filtrage est exécuté après déclaration de l'environnement d'exécution. Le choix d'un "package" déterminé par l'utilisateur permet l'exécution d'un algorithme plutôt qu'un autre. Il peut même redéfinir son propre algorithme ... à ses risques et périls.

Deux types de lissage ont été traités, basés sur le fait qu'une arête avait une largeur minimale autorisée. Dans le premier cas, lorsqu'une arête est inférieure à cette valeur minimale, elle disparaît au profit de l'arête supérieure qui lui est contiguë. Dans le second cas, c'est au profit de l'arête inférieure qui lui est contiguë (FIGURE F-10). Dans les deux cas, il y a surface perdue, et possible détection d'erreur au niveau du DRC. Cette détection est résolue par une observation des masques dans la fenêtre correspondant à la zone de violation.

Lors de l'appel d'une cellule, un utilisateur doit pouvoir choisir, suivant le contexte, de spécifier le type de frontière avec lequel il souhaite travailler.

Lors d'une opération d'assemblage il peut le faire en spécifiant:

(Frontiere appel paramètres)

Lors du processus d'assemblage, les paramètres associés à l'appel sont pris en compte à l'instant du calcul dynamique

de la frontière avec l'attitude d'instanciation. Le lissage choisi est effectué.

La détection d'erreurs imaginaires au niveau du DRC peut ainsi être levée, et les erreurs de mise à la garde rectifiées lorsqu'elles ont un impact important sur la qualité de l'assemblage. Le principal défaut, de cette méthode, est que le temps d'exécution des algorithmes se trouve considérablement ralenti.

5.3.2.4 - Mise à la garde de deux frontières

L'algorithme de mise à la garde de deux frontières sert essentiellement à l'assemblage de deux cellules, et se fait à l'aide de deux paramètres, un delta technologique, et la distance minimale entre les cellules. Cependant, lors d'un assemblage, l'algorithme utilisé n'est pas tout à fait le même, car aux frontières s'ajoutent les ombres qui permettent la violation de gardes là où un simple algorithme sur les frontières détecte violation. Le principe est le même, seule une clause est rajoutée, se référant au type de l'arête traitée (frontière ou ombre).

Tout d'abord, les deux cellules traitées sont placées le long d'un axe (x par exemple). L'un est fixe (A) et l'autre mobile (B). Le premier positionnement est effectué de façon à ce que les enveloppes des deux cellules ne se recouvrent pas. Il en découle que pour toute arête de la cellule mobile, le calcul de sa distance aux arêtes correspondantes de la cellule fixe est toujours de même signe (FIGURE F-11). La distance choisie, est calculée. Elle est le minimum de toutes les distances de violation de garde. Il y a une

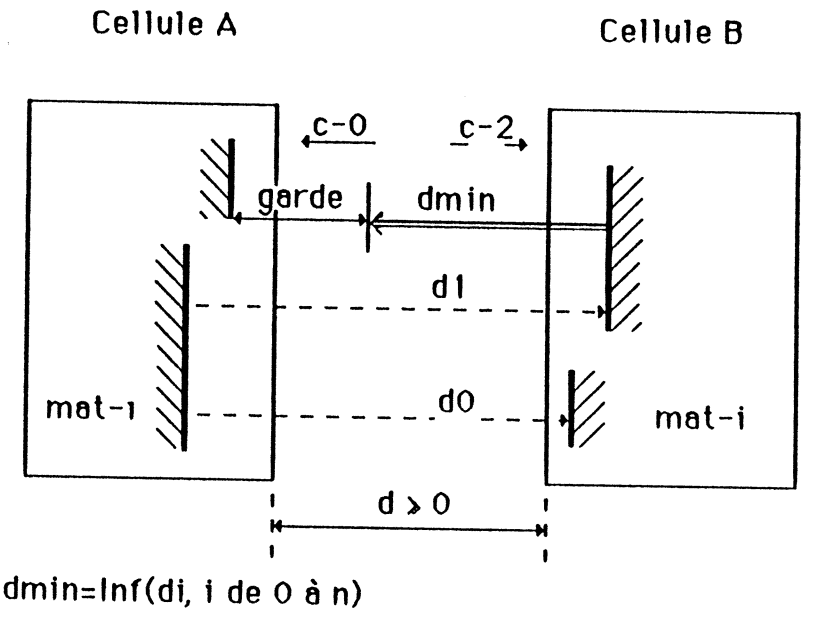


FIGURE F-12

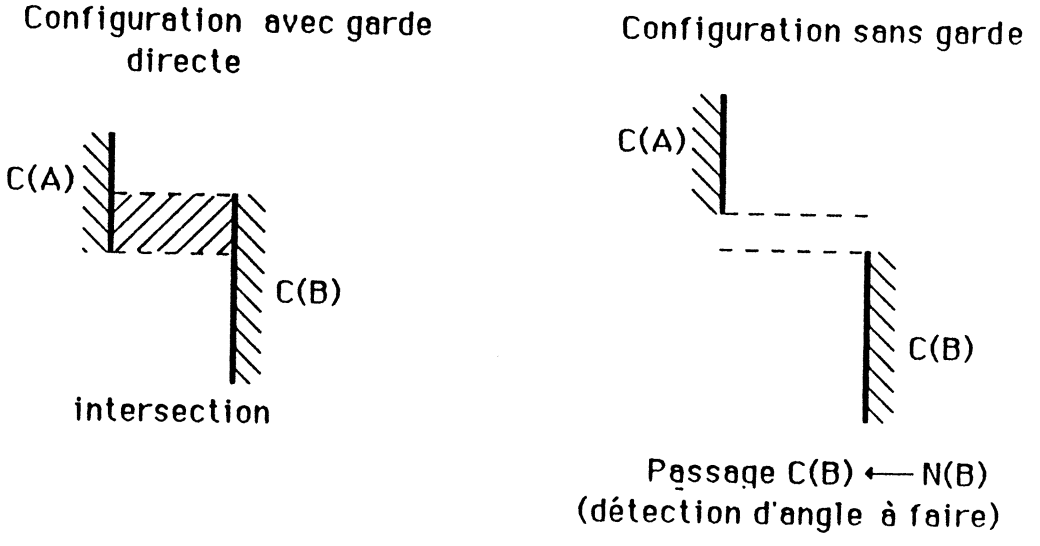


FIGURE F-13

fonction qui calcule cette distance à partir du tableau technologique.

Sur chaque niveau technologique de A, en fonction de la table des règles technologiques, la distance maximale dont on peut rapprocher la cellule B est calculée (Exemple pour un NMOS: cas de la matière poly, il existe une règle poly/diff, valeur 1, et une règle poly/poly, valeur 2, pour des règles en lambda).

La valeur de rapprochement de B est alors "dmin", avec:

dmin=Inf(dmin(mat-i), i de 1 à n)

Le calcul du dmin(mat-i) se fait en prenant les deux listes d'arêtes c-j(A) et c-(j+2)(B), et en calculant de façon récursive le dmin.

Pour deux arêtes opposées, il est possible de calculer leur distance. Il suffit alors de retirer la garde de la règle en cours, pour obtenir la distance de rapprochement possible entre les deux arêtes (FIGURE F-12). Lors d'un assemblage, il y a deux types d'arêtes, les arêtes obstacles, qui vérifient les règles technologiques, et les arêtes traversables qui correspondent à la déclaration d'une ombre d'accès à la cellule, ou bien une ombre de transparence sur la cellule. Le positionnement est alors effectué, soit par des coordonnées de point de connexion, soit par le descriptif d'une ombre.

Le parcours récursif des deux listes d'arêtes se traduit par une propagation du "dmin" le long des deux listes.

Cas de détection d'angle

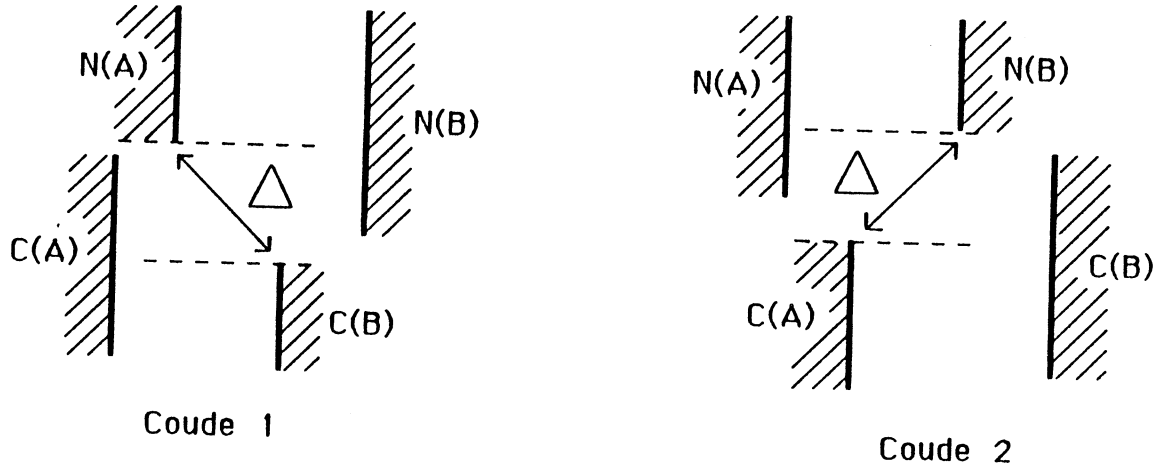


FIGURE F-14

Assemblage de A et B suivant c-0

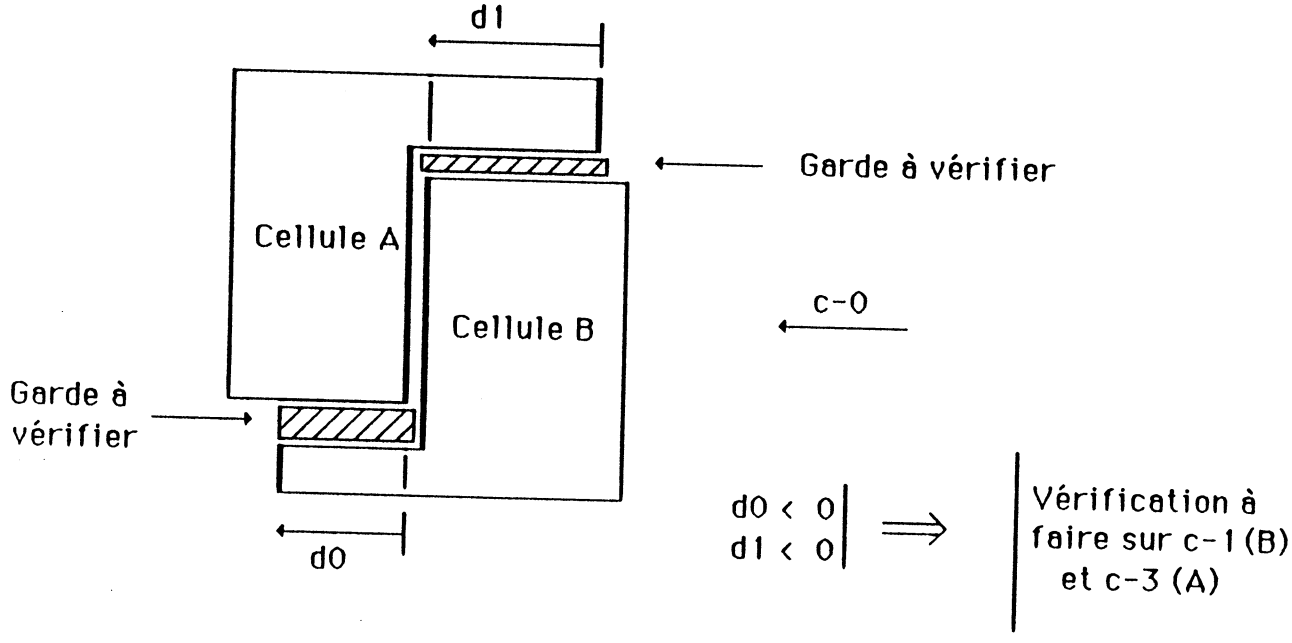


FIGURE F-15

Si on considère la liste L(A) et la liste L(B), les arêtes courantes C(A) et C(B), ainsi que les deux arêtes suivantes des deux listes N(A) et N(B), à l'initialisation C(*) et N(*) sont initialisées avec les deux premières arêtes de L(*).

Les positions relatives des deux arêtes traitées sont déterminées. Il peut y avoir intersection ou non (FIGURE F-13). Les actions sont alors les suivantes:

-Détection d'une garde entre C(A) et C(B), si non passage à l'arête suivante, si oui, calcul du nouveau dmin(mat-i) et passage à la suivante.

-Le passage à l'arête suivante commence par le test sur la liste à incrémenter, puis il cherche si la configuration C(A), N(A), C(B), N(B), présente un coude (FIGURE F-14). Il y a deux types de coudes. Si un coude est détecté, il y a calcul de la distance (dc) de garde admise, et un nouveau calcul de la distance "dmin" est effectué: dmin=Inf(dmin,dc).

-Ensuite, il y a exécution de l'incrément sur C(*) et N(*). L'opération s'arrête lorsque une des deux listes L(A) ou L(B) est arrivée à son terme.

Lorsque l'opération a été effectuée sur tous les niveaux technologiques, la cellule B est décalée de "dmin" le long de l'axe de mise à la garde. Tous les points sont recalculés.

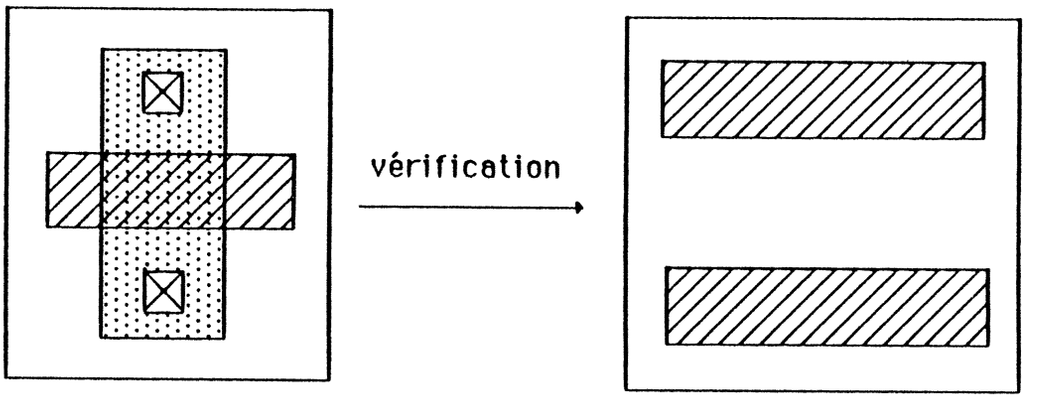


FIGURE 0-0

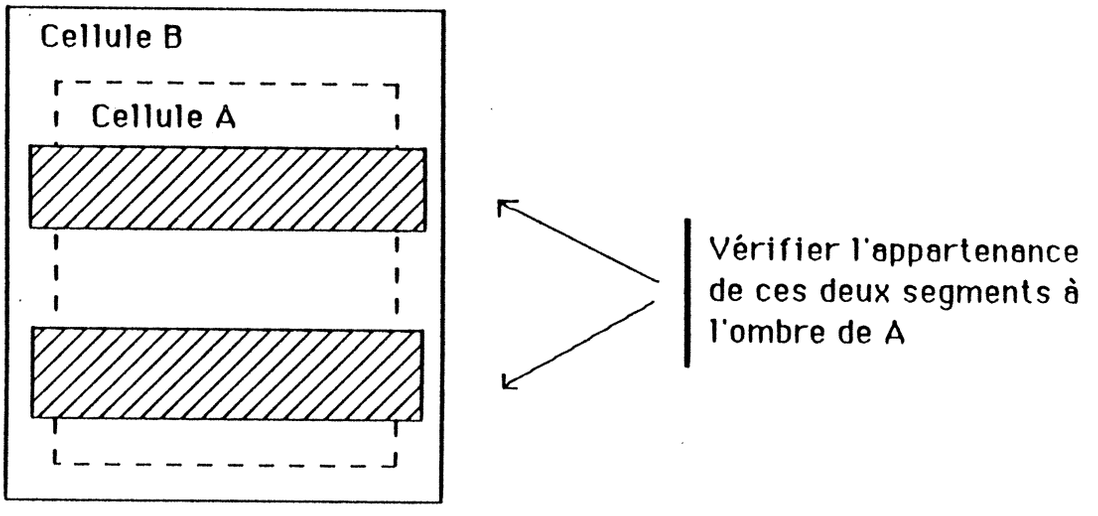


FIGURE 0-1

Cette opération utilise un algorithme général, indépendant de la technologie, et de la direction traitée. Cependant, il convient de remarquer qu'elle n'assure pas la cohérence du positionnement. En effet, si la distance de rapprochement introduit le fait que des arêtes du côté "c-j(A)" aient une distance négative à celles du côté "c-(j+2)(B)", alors une vérification des gardes sur les côtés "c-(j+1)" et "c-(j-1)" est nécessaire (FIGURE F-15).

5.4 - Les ombres

5.4.1 - Principe

Elles représentent les possibilités de violation de l'espace de la cellule (9-1,146), lors d'instanciations de celle-ci. Elles dépendent donc de contextes qui peuvent être différents. Leur intérêt est de contrôler ces violations sans avoir à faire des vérifications de règles de dessin de caractère hiérarchique, extrêmement coûteuses. Le concepteur, en déclarant de manière explicite les divers contextes d'appel d'une cellule lors de sa conception permet d'assurer la justesse et la cohérence de son dessin. Les travaux de vérification sont alors relativement simplifiés, car structurés.

Dans un premier temps, il convient de vérifier que la cellule construite est bien en accord avec les contextes déclarés, et qu'il n'existe pas d'erreur de conception au niveau des règles de dessin (violation de garde). Ceci est un problème hiérarchique à un niveau (FIGURE 0-0). Il peut être traité de façon extrêmement simple par unification

Méthode d'accès au corps d'une cellule

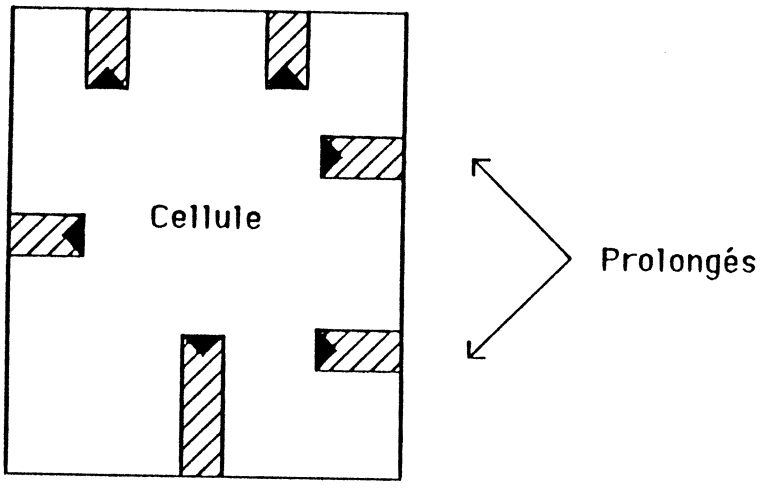


FIGURE 0-2

La cellule B appelle l'ombre A

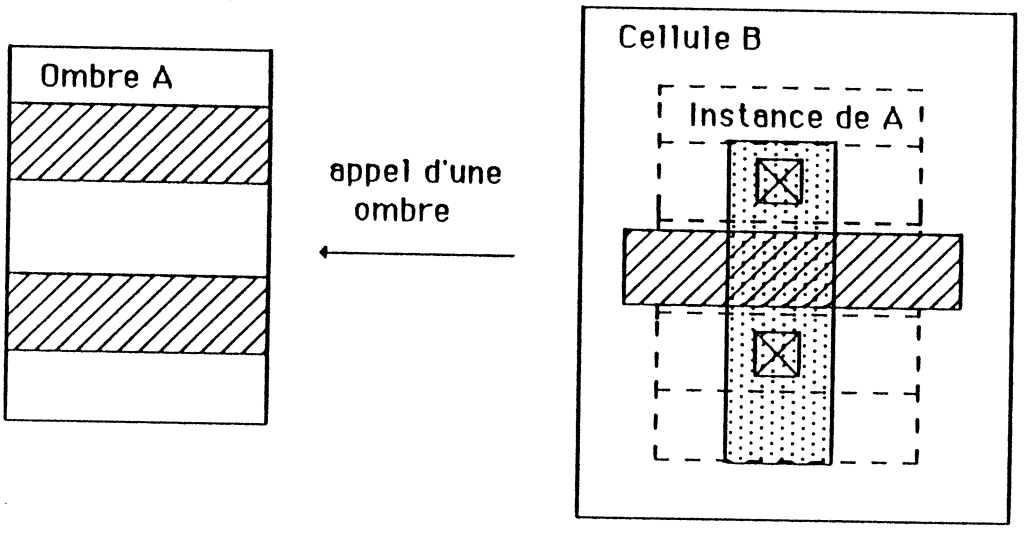


FIGURE 0-3

dynamique et temporaire des deux niveaux.

Ensuite, lors de l'instanciation d'une cellule ayant une ombre déclarée, il suffit, à chaque violation de l'espace de cette cellule, de vérifier que l'élément entrant en violation est en conformité avec l'ombre déclarée dans la cellule appelée (FIGURE 0-1).

5.4.2 - Choix d'implantation

L'implantation des ombres peut être faite de diverses façons.

F.Anceau a proposé, en 1984, d'associer tous les symboles de connexion à la cellule sous forme de segments rectilignes accrochés aux points de connexion de la cellule et rejoignant ses bords (FIGURE 0-2). Ce sont les prolongés. Les ombres, pour leur part, représentent les contextes d'appel d'une cellule sous forme de cellules ombres. La cellule se contente alors d'instancier une ombre avec une attitude donnée (FIGURE 0-3). Le gros avantage de cette méthode est que le maximum d'information est partagé, et les données sont indépendantes. Lors d'une vérification hiérarchique, on n'a besoin que de la cellule ombre. Il est évident que dans une approche "bit-slice", il y a peu de cellules ombres (un bus suffit), et les vérifications hiérarchiques sont très performantes.

Dans le cas traité, la déclaration des ombres est rattachée au corps des cellules. Le partage des ombres par les cellules n'est pas possible. Ceci n'est pas très important, car l'utilisation essentielle est la vérification hiérarchique des règles de dessin. Lorsque on sait faire un

Passages admis sur une cellule

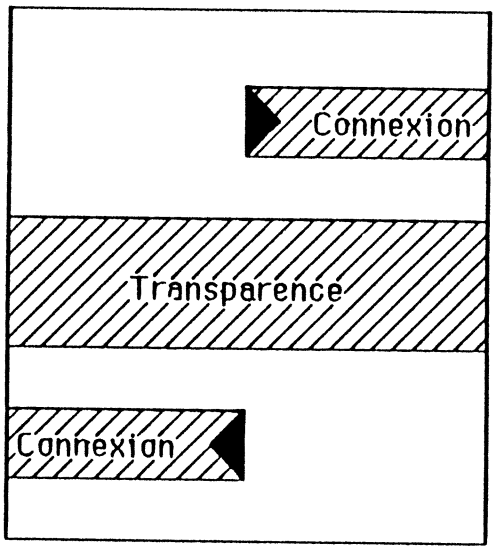
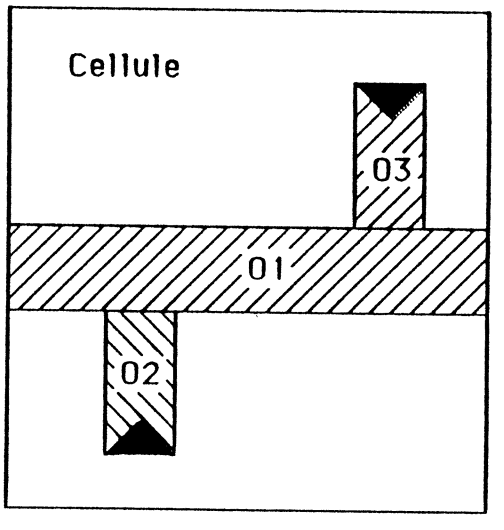


FIGURE 0-4

Ombre - Objet hiérarchisé



Ombre principale: 01
Ombres secondaires : 02,03

02 et 03 peuvent être déclarées exclusives.

(si une connexion est réalisée avec l'une d'entre elles, l'autre devient inaccessible)

FIGURE 0-5

"DRC" sur deux niveaux, la possibilité de déclarer des cellules opaques (dont le contenu n'est pas visible) doit permettre d'implanter ce mécanisme à postériori. Il ne conditionne pas la structure du système.

Il convient de voir l'ensemble des possibilités de déclarations au niveau d'une cellule, les types d'ombres admis, sachant qu'ils ont été conçus dans le but d'assemblages justes par construction, et afin de palier à certains problèmes rencontrés dans les représentations symboliques (partage d'éléments et vérification des règles de dessin).

5.4.3 - Définition des ombres

A partir d'ici, les ombres ont un caractère restrictif. Est appelé "OMBRE", l'ensemble des éléments appartenant à une cellule et qui décrivent les divers passages admis sur cette cellule lors d'instanciations (FIGURE 0-4). Une ombre doit être cohérente vis à vis de la cellule dans laquelle elle est déclarée. Elle doit respecter les règles de dessin corespondant à la technologie employée. Le seul moyen de se connecter à la cellule, est par la matérialisation d'ombres de connexion.

Nous distinguons deux types d'ombres, les ombres de connexion et les ombres de transparence (FIGURE 0-4).

5.4.3.1 - Ombres de connexion

Elles sont un chemin d'accès à la cellule. Elles référencient les points de connexion. Ces points sont

référencés par au moins une ombre. Un point qui ne l'est pas, disparaît. Ce qui veut dire que tout point de connexion a été déclaré à un moment ou un autre. La connexion implicite n'existe pas, et tout point de connexion possède toutes les informations que l'on est en droit d'attendre de sa part (coordonnées, nom, typage, ...).

5.4.3.2 - Ombres de transparence

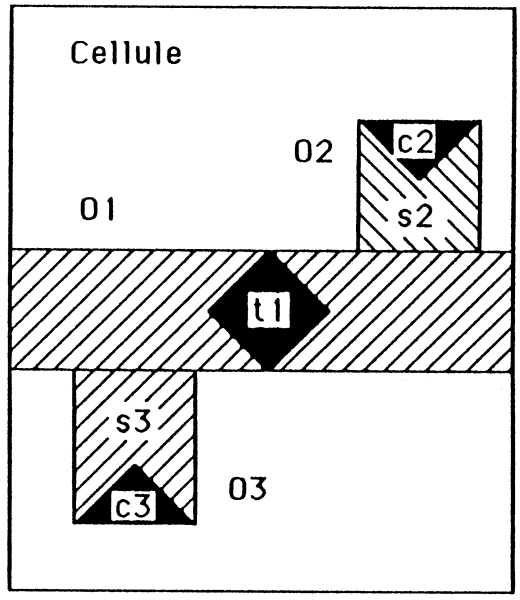
Elles sont un chemin à travers la cellule, sans connexion aucune avec un des éléments de la cellule. Il est possible de traverser la cellule à un niveau de hiérarchie différent, tout en conservant une cohérence de dessin et en ne modifiant pas le graphe électrique (il demeure cependant le problème des couplages parasites liés à des effets contextuels et dynamiques. Il est possible de le résoudre, moyennant un coût de calcul. Cependant, le problème nous semble plutôt être au niveau des simulateurs et des moyens de détection des couplages). Les ombres de transparence sont explicites, à l'exception des passages directs qui peuvent être implicites moyennant un coût de vérification (détection d'un passage à un niveau technologique de frontière donné, absence d'interaction avec d'autres niveaux de frontière, et destruction éventuelle d'ombres existantes).

5.4.3.3 - Ombres et paramétrage

Les ombres, telles qu'elles sont conçues, doivent permettre d'accéder aux points de connection d'une cellule, de manière paramétrée. En effet, l'étude de circuits générés par assemblage montre la limitation du simple accès par prolongés, dans la mesure où la création de fonctions lors

Décomposition d'une ombre

Ombre : - arêtes de sortie
- canal
- ombres secondaires / ombre mère
- accès à un connecteur
- ombres exclusives
- attributs



Ombres : - Principales :	01	t1	(transparence)
		02, 03	(filles)
		clock	(type)
		01	(nom)
- Secondaires :	02	c2	(connexion)
		s2	(corps)
		01	(mère)
		⌉03	(exclusive)
		02	(nom)

FIGURE 0-6

d'un assemblage est interdite (un assemblage n'est qu'une union de graphes électriques). Pour pallier à ce problème, les ombres sont considérées comme des objets hiérarchisées (FIGURE 0-5), répondant à un certain nombre de messages (ombre de connexion, de transparence, liste des terminaux, ombres complémentaires, ...).

5.4.4 - Structuration des ombres

La décomposition de l'ensemble de la structure est la suivante (FIGURE 0-6):

-liste des points de connexion

-liste de canaux de passage

-liste d'arêtes.

5.4.4.1 - Points de connexion

Les points de connexion sont les articulations de la cellule sur lesquels on désire venir se connecter. Ils sont structurés sous la forme d'une A-liste **(matière liste-arts)**.

5.4.4.2 - Canaux d'accès

Les canaux sont les chemins d'accès à la cellule. Ils sont de même nature que les symboles de base permettant des interconnexions entre éléments actifs. Un canal est rattaché au bord d'une cellule par des arêtes, et au corps par le rattachement à des points de connexion.

5.4.4.3 - Arêtes

Les arêtes sont les points de pénétration dans la cellule.

Une arête correspond à un élément donné de la frontière étant au même niveau technologique.

5.4.4.4 - Décomposition des ombres

Les ombres sont décomposées de la façon suivante:

-ombres principales

-ombres secondaires

-accès à un connecteur

5.4.4.4.1 - Ombre principale

Une ombre principale possède un corps (stick), sur lequel vient se rattacher un certain nombre d'attributs, parmi lesquels des arêtes de sortie, des ombres secondaires, des ombres exclusives ou des accès à un connecteur.

5.4.4.4.2 - Ombre secondaire

Une ombre secondaire est "un bras" d'une ombre principale. Ce sont les ombres secondaires qui permettent le paramétrage par leur sélection ou leur non-sélection, exemple 02 ou 03 dans (FIGURE 0-6). Elles font référence à une ombre principale, possèdent des exclusives et des accès à un connecteur.

5.4.4.4.3 - Accès à un point de connexion

L'accès à un point de connexion est une référence à un point de connexion de la cellule, auquel s'ajoute l'état d'expansion de ce point, ceci afin de permettre de manière hiérarchique une expansion correcte des inter-connexions

Expansion des points de connexion

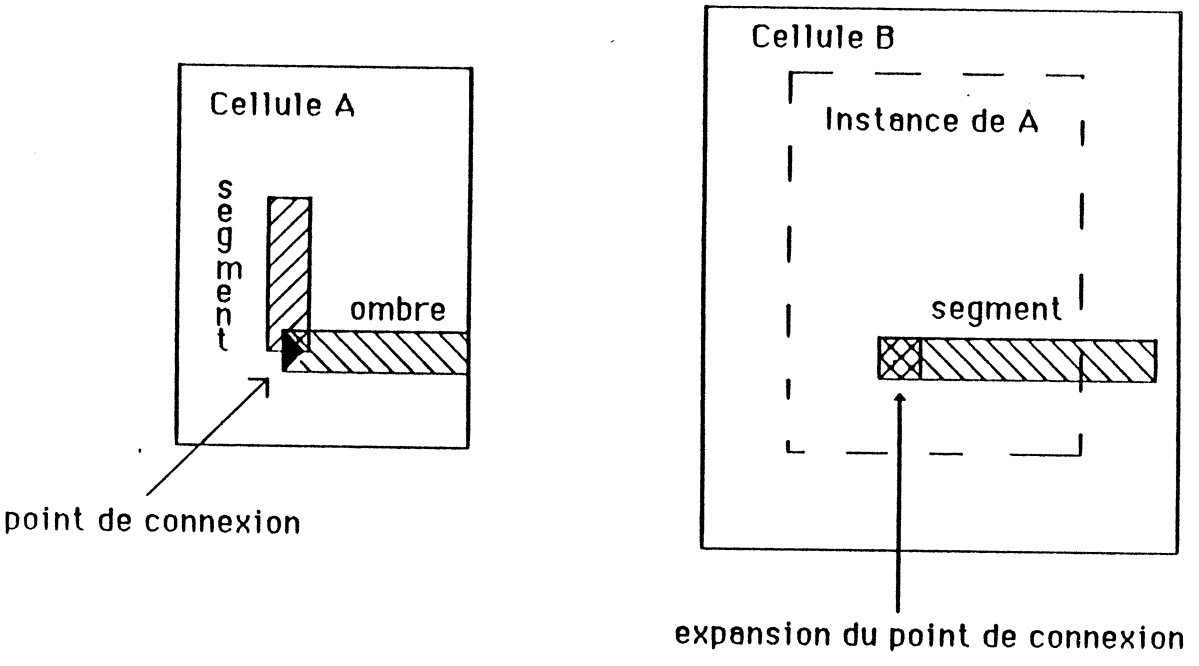


FIGURE 0-7

Ombres de la cellule B

Descriptif de 00 : ((mat-i (a-0 (Y1 . Y2)))
(extensions (01 02))
(attributs ((nom 00)))

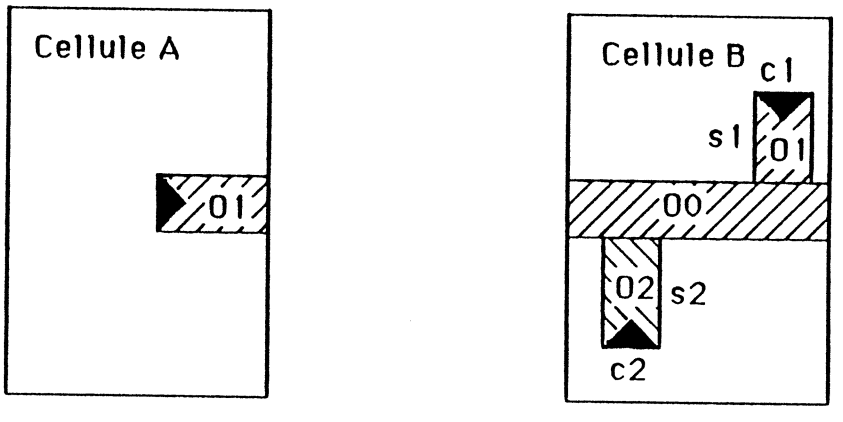


FIGURE 0-8

(FIGURE 0-7).

5.4.4.4.4 - Structure de l'ombre

Les ombres référencent les arêtes par niveau technologique, lorsqu'elles ont une sortie. Une ombre peut alors être décrite sous forme de la structure lisp suivante (clef, valeur), sachant que l'objet a une structure dynamique, et que les champs (clef, valeur) ne sont créés que lorsqu'une valeur leur est affectée (FIGURE 0-8):

```
((sorties       liste-d'arêtes)
 (corps         éléments-stick)
 (connexions    liste-accès)
 (exclusifs     liste-ombres)
 (extensions    liste-ombres)
 (rattachement  ombre)
 (attributs     liste-attributs)
 (numero        entier))    ; il s'agit du pointeur
```

Les sorties sont structurées sous la forme:

```
((mat-1 s-1) ... (mat-i s-i) ... (mat-n s-n))
```

Avec, s-i :

```
((c-0 liste-arêtes)
 (c-1 liste-arêtes)
 (c-2 liste-arêtes)
 (c-3 liste-arêtes)
 (a-0 transparence)   ; a-0 axe des x
 (a-1 transparence))  ; a-1 axe des y
```

Il est bien évident que dans presque tous les cas les ombres se réduisent à quelques champs. De ce fait les algorithmes

Partage d'élements entre cellules

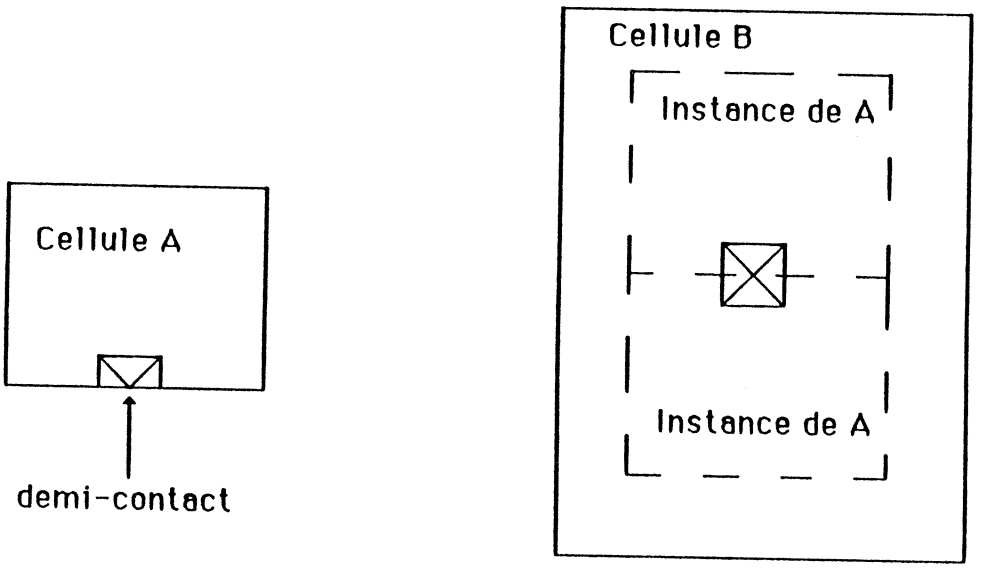
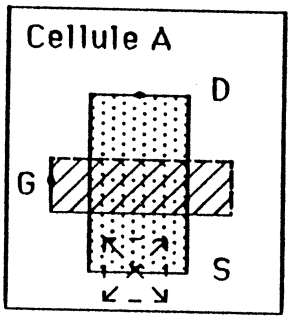


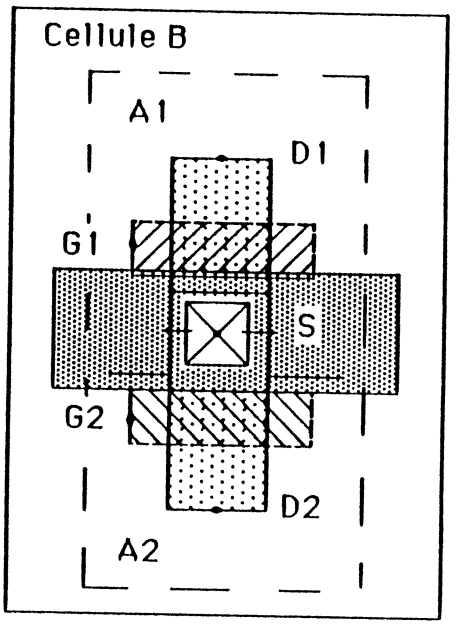
FIGURE 0-9

Partage d'un contact

Contact déclaré sous forme d'une ombre dans A

Contact physiquement réalisé dans B





L'articulation S de B a pour propriétés: un contact mat1/mat2 et deux chemins d'accès au point de connexion S de A par les deux instances A1 et A2.

FIGURE 0-10

privilégient les cas simples. L'utilisation d'ombres complexes est autorisée, mais le traitement est alors beaucoup plus long. La raison est que sur ces structures, les fonctions de manipulation et d'édition ont été conçues les plus générales possibles, en utilisant pour cela la puissance de lisp sur le traitement des listes, et le calcul dynamique des fonctions à appliquer. A partir du moment ou sont traitées des ombres complexes, il y a répétition du calcul des sémantiques à appliquer. Même si par un typage des données et l'utilisation des "packages" lisp, on arrive à structurer et rendre performante cette succession d'actions, sa répétition la ralentit considérablement.

Il est possible de conclure en disant que des mécanismes complexes peuvent être facilement implantés, moyennant des logiciels et des structures de données relativement simples. Cependant, l'utilisation de cette possibilité de complexité n'est pas conseillée en dehors des cas ou il n'est pas possible de faire autrement.

5.4.5 - Symboles partagés entre cellules

5.4.5.1 - Problème

Dans la conception traditionnelle de circuits, il est usuel de partager des éléments ayant une fonction électrique donnée entre plusieurs cellules. Ce n'est qu'au cours de l'assemblage que ces élements apparaissent (FIGURE 0-9). Ceci présente un certain nombre de problèmes dans le cadre d'une représentation symbolique, car le partage de symboles entraîne des pollutions sur les cellules. Ceci est traduit

par la fusion d'arêtes lors de la composition de graphes (par assemblage), alors que les assemblages sont conçus comme des fusions de noeuds des graphes. Il est possible d'admettre que les cellules peuvent partager des éléments, mais alors, si le recouvrement de cellules est autorisé, la gestion des frontières et des algorithmes de mise à la garde se complexifient énormement. Les frontières n'ont plus de raison d'être.

5.4.5.2 - Solution

Une solution simple à ce problème est l'utilisation des ombres. Elles ont alors le rôle d'éléments paramétrés réalisés en fonction du contexte d'instanciation de la cellule. Il n'y a pas recouvrement de cellules, et tous les mécanismes utilisant les frontières peuvent être considérés comme fiables pour la génération algorithmique de circuits.

Les élements qui sont partagés entre cellules sont déclarés sous forme d'ombres. Il ne peut s'agir d'élements actifs, ceci est interdit en raison de la méthodologie de conception (les fonctions sont au niveau des feuilles, et non dans la hiérarchie). Lors de l'instanciation d'une cellule, les ombres qui se trouvent dans un contexte de résolution sont physiquement réalisées, et le noeud électrique qui en découle correspond réellement à ce qu'a fait le concepteur (FIGURE 0-10). L'expansion des éléments est faite conformément au descriptif des points de connexion correspondant. Les expansions obtenues sont vérifiées comme admises lors de la déclaration des ombres.

5.4.5.3 - Règle d'exclusion

Lors d'un assemblage, le mécanisme de partage des éléments est réalisé suivant une règle d'exclusion extrêmement simple. Lorsque deux ombres, appartenant à deux cellules assemblées se superposent, tous les élements communs aux deux ombres sont réalisés, les autres sont éliminés. Il n'a plus de problème de "DRC" hiérarchique, de pollution sur les cellules et de création de fonctions implicites ou parasites.

5.4.5.4 - Intérêt

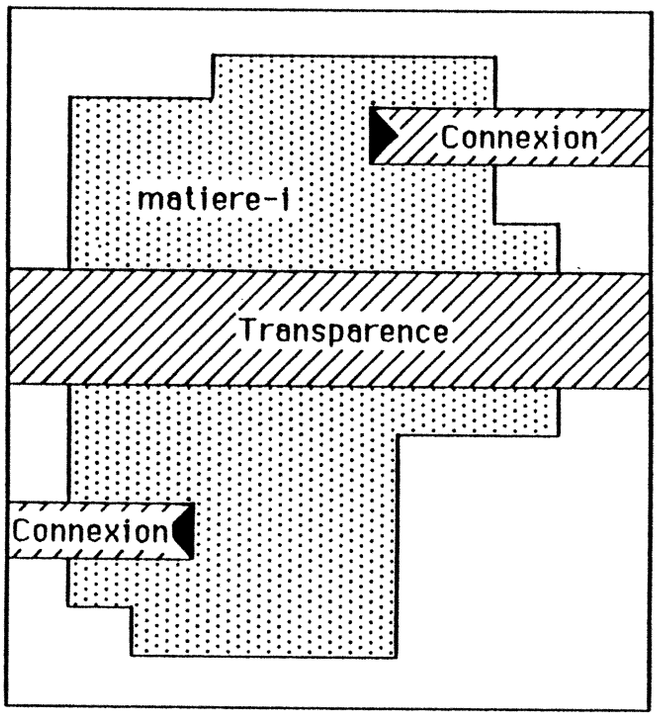
Le choix de ce mécanisme fait apparaître une plus grande puissance de création et de structuration pour le symbolique. La perte d'espace est minimale pour l'assemblage. Il est possible de réaliser des morceaux de circuits ayant une structure régulière avec une densité aussi grande qu'avec des systèmes purement manuels. L'observation a été faite sur les circuits FELIN (54) et SCALA (70).

D'autre part, un "DRC" hiérarchique performant est ainsi conservé. Il se contente de vérifier la conformité des instances par rapport aux modèles déclarés, et l'extraction électrique où les parcours de graphes électriques demeurent triviaux.

5.5 - Récapitulatif

En conclusion, il apparaît que la vision externe d'une cellule (dans le cadre d'une conception algorithmique de circuits par assemblage de cellules, avec pour

VISION EXTERNE D'UNE CELLULE



et les attributs

- nommage
  - cellules
  - ombres
  - connexions ...
- typage
  - ombres
  - connexions ...

FIGURE 0-11

caractéristiques une grande densité, l'utilisation de mécanismes simples, et un respect contrôlé et automatique des règles de dessin), pour une cellule, se traduit par:

-une Bbox (rectangle)
-des frontières
-des ombres de connexion, de transparence
-des attributs (nommage, typage, ...)

(cf. FIGURE 0-11)

6 - Construction algorithmique de circuits intégrés par assemblage de cellules

6.1 - Présentation

Un assembleur de cellules peut être conçu en plusieurs parties:

-un éditeur de cellules feuilles qui représentent une fonctionnalité donnée, et sont constituées de quelques dizaines de transistors au plus. L'éditeur doit permettre de dessiner des cellules denses avec une perte de temps minimale, d'où l'idée d'un éditeur symbolique métrique avec contrôle de garde incrémental, expansion incrémentale des élements manipulés (sticks), et maintien d'un graphe électrique cohérent.

-un ensemble de mécanismes, qui à partir d'une cellule métrique, arrive à en déduire une vision externe, sans se préoccuper du contenu. Ce sont les mécanismes de nommage,

de déclaration de connecteurs, d'équipotentielles, d'extraction des frontières.

-un ensemble de mécanismes de composition qui permettent, à partir d'une spécification, de générer des cellules composées métriques. Ce sont les mécanismes de placement, routage, remontée des informations.

-un ensemble de fonctions immergées dans un langage de haut niveau, qui permettent la manipulation des données, l'exécution des actions, la description de circuits sous forme de programmes.

Les problèmes essentiels sont les calculs de mise à la garde et la détermination des interconnexions, ceci pour obtenir un système puissant et agréable. Les choix effectués sont l'écriture de l'ensemble des mécanismes et la spécification de fonctions d'assemblage en Le_Lisp. Les mécanismes d'assemblage sont conçus pour évoluer en fonction des besoins rencontrés.

6.1.1 - Différents types d'assemblage

Lorsqu'un concepteur est amené à concevoir un circuit, il peut se trouver devant plusieurs cas de figure.

6.1.1.1 - Routage global

Lorsque le concepteur a en mains un ensemble de blocs connus, son problème est un problème de routage global. Des solutions nombreuses et variées existent à ce problème. La base de la plupart d'entre elles est la résolution d'un graphe polaire ou de contraintes, et l'utilisation d'un routeur bi-couches (2,47,59,71,80,88,105,134,179,189).

6.1.1.2 - Assemblage de cellules déformables

Un autre cas, rencontré dans les systèmes CAO où le concepteur a la possibilité d'utiliser des cellules "stretchables", est la composition de cellules par aboutement en fonction de leur fonctionnalité. A une fonctionnalité donnée correspond une cellule. Le concepteur n'a qu'à formaliser les fonctionnalités qu'il souhaite relier, ou assembler. Il en découle un graphe de relations entre cellules, sachant que chaque cellule possède ses propres contraintes (position relative des points de connexion). La composition des cellules est conçue par simple aboutement et "stretch" des cellules. Ceci est traduit par un graphe de contraintes en X et en Y. Dans cette approche, il convient de noter que la représentation hiérarchique est particulièrement bien adaptée. L'utilisation de méthodes de description récursives des circuits l'est également (116). Il est relativement simple de traduire les contraintes liées à une cellule de composition en une liste de contraintes sur les connexions externes de la cellule. Il y a réduction des contraintes dans une démarche "Bottom-Up", et résolution de celles-ci dans une démarche "Top-Down" récursive (100).

Cette démarche est relativement agréable, et permet la conception rapide de circuits à partir d'une description de haut niveau (100). Cependant, elle présente un certain nombre d'inconvénients. Lors de la résolution des contraintes, on est amené à déformer plus ou moins les cellules de base, allongeant un certain nombre de "fils", et modifiant par la même occasion leurs performances électriques, ou leur comportement dans le cas de fonctionnement dynamique. Un autre point important est la quantité de silicium perdue par de tels systèmes. Elle peut

aller jusqu'à deux ou trois fois la surface utilisée par d'autres assembleurs (MacPitts,Apollon) (77,152).

6.1.1.3 - Assemblage métrique

Le seul type d'assembleur qui est vraiment performant, est l'assembleur par aboutement de blocs métriques. Tout l'effort de conception du dessin est à la charge du concepteur. Il doit, dans un premier temps, déterminer quelle est la cellule feuille la plus contraignante. Il doit ensuite adapter toutes les cellules utilisées au pas de cette cellule, avant de concevoir son algorithme de composition. L'expérience montre que cette méthode est loin d'être rébarbative. Tous les circuits présentant des structures régulières aux contraintes électriques importantes sont conçus ainsi. Le niveau de performance qui est demandé, aux machines que l'on intègre, est tel que l'architecture de leur partie opérative est dessinée à la main (9). Le problème posé à la machine est celui de la réalisation d'un algorithme de composition.

En revanche, dans certains contextes où les contraintes sont moins fortes, des assembleurs avec résolution de contraintes peuvent être utilisés avec efficacité. C'est le cas notamment au niveau des interfaces de gros blocs, là où il y a toujours perte de place et où les contraintes électriques sont parfaitement maîtrisées.

6.1.2 - Principe d'assemblage

Les mécanismes réalisés utilisent tous les mêmes données, à savoir les cellules métriques générées par l'éditeur, ou toute cellule ayant la vision externe spécifiée dans le

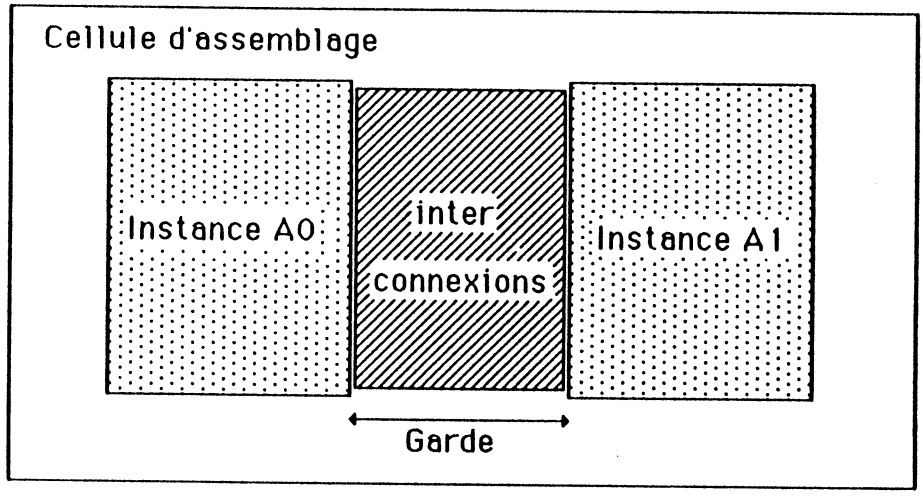


FIGURE A-0

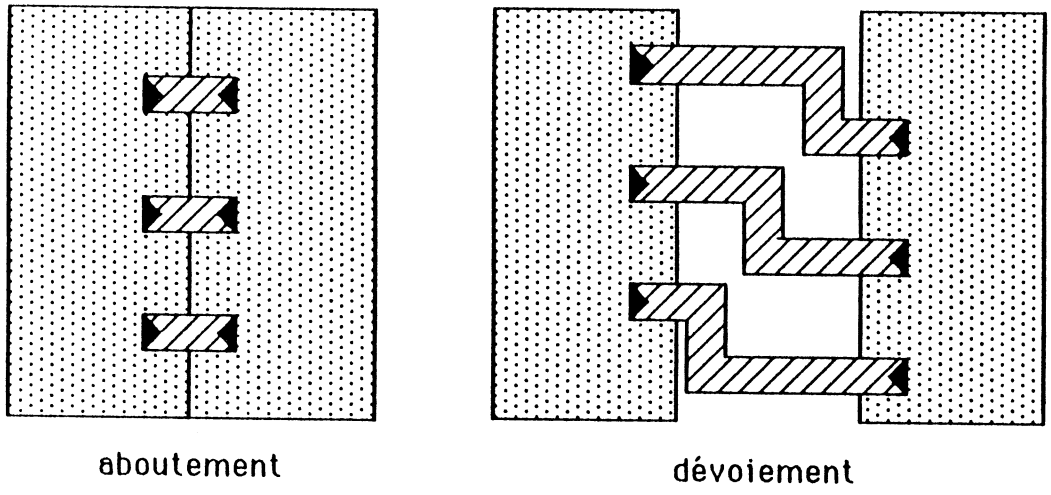


FIGURE A-1

chapitre précédant.

Les objets manipulés sont:

-la cellule

-l'appel de cellule

-l'ombre

-le point de connexion

-le nom (nom de connexion, appel, cellule, ombre)

Le mécanisme d'assemblage est extrêmement simple. Il consiste à prendre deux cellules, à les instancier dans une cellule d'assemblage, relativement l'une par rapport à l'autre, à générer une inter-connexion, et à les placer en respect des gardes liées à la technologie employée (FIGURE A-0).

6.1.3 - Opérateurs d'assemblage

Il est possible de diviser les opérateurs d'assemblage en trois familles: l'aboutement, le dévoiement, le routage bi-couches.

6.1.3.1 - L'assemblage direct

L'assemblage direct ou "aboutement" (Figure A-1) se caractérise par le fait que les connexions ne font perdre aucune place. Ce sont les gardes technologiques entre les deux cellules assemblées, traduites par le calcul du "dmin" entre frontières, qui donnent les positions relatives des cellules assemblées.

6.1.3.2 - L'assemblage par dévoiement

L'assemblage par dévoiement (FIGURE A-1) est assez proche de l'assemblage direct dans la mesure où l'ordre des connexions reste le même. Cependant, le calcul des positions relatives des cellules nécessite le calcul du nombre de coudes à générer. La position relative devient la somme du positionnement par garde sur les frontières et d'une fonction proportionnelle au nombre de coudes réalisés. Une autre différence notable est que dans le cas de petites cellules, la surface de dévoiement devient plus grande que la taille des cellules elles-mêmes. L'utilisation de ce routage n'est donc pas toujours intéressant. D'autre part, dans le cas d'un routage par dévoiement avec de nombreux points de connexion et de nombreux coudes, il apparaît que l'utilisation d'un routage bi-couche, au problème électrique près, devient plus performante. Le gain de place est non négligeable. En conséquence, le dévoiement connaît un nombre d'applications limité au deux extrêmes (trop grande ou trop petite complexité). Son utilisation essentielle est incontestablement l'adaptation des sorties d'une cellule. Par dévoiement les points de connexion sont espacés régulièrement ou mis sur grille. Le nombre de coudes généré est faible, la surface perdue également.

6.1.3.3 - L'assemblage par routage

Le dernier type de routeur, le routeur bi-couches, est utilisé pour permettre la réalisation de connexions complexes entre cellules (FIGURE A-1). Il peut souvent être traité comme une cellule d'interconnexion. Le problème du placement des deux cellules assemblées se traduit en un double assemblage direct avec une cellule de routage.

Les principaux inconvénients de ces routeurs sont la

difficulté à maîtriser les contraintes électriques appliquées aux différents signaux à router (contraintes traduites par des longueurs maximales de fils ou largeurs différentes). Contrairement au cas du dévoiement ou les algorithmes s'appliquent parfaitement avec de bonnes performances (84), ici les algorithmes sont beaucoup moins performants et de plus , bien souvent ils tolèrent très peu le guidage par l'homme, ou l'aide à la décision. C'est pourquoi de nombreuses méthodes de routage bi-couches sont développées (47), les dernières réalisations se rapprochant de l'intelligence artificielle (80).

6.2 - Immersion Lisp des mécanismes d'assemblage

Tous les mécanismes sont conçus dans le cadre d'une programmation orientée objet. Le concepteur exécute des procédures sur ces objets, envoie des messages aux objets, qui en fonction de leur contexte choisissent les actions à accomplir.

En conséquence, toutes les fonctions de manipulation des objets ont pour paramètres le contexte d'exécution, l'objet et les paramètres annexes. L'évaluation d'une fonction lisp, correspondant à une action sur un objet, retourne toujours le pointeur sur cet objet. Le concepteur ne manipule donc que les pointeurs, ou les listes de pointeurs sur les cellules, les appels, les connecteurs, les noms, les ombres.

6.2.1 Méthode

La méthode d'assemblage est relativement proche de LUBRICK (146). Il y a ouverture de cellule d'assemblage, ajout d'appels et fermeture de la cellule. Une autre approche peut

être employée, qui permet de concevoir l'assemblage sous forme d'arbres binaires. Cette méthode s'applique bien au cas d'implantation d'algorithmes sur silicium (142). Cependant, elle n'est considérée que comme un cas particulier du système. Une cellule est ouverte. Deux cellules sont assemblées. La cellule est refermée. La réalisation d'un tel assembleur peut être faite à l'aide d'un ensemble de macro-fonctions.

6.2.2 - Contexte d'assemblage

Lors de l'évaluation d'une procédure d'assemblage, il existe un contexte courant qui est constitué par la liste des hiérarchies présentes, la technologie, un routeur par défaut, une hiérarchie courante, une liste d'outils sélectionnés (DRC, expanseur, ...).

L'ouverture d'une cellule d'assemblage se traduit par la fonction:

(open-fig 'X)

Dans laquelle X représente un certain nombre de paramètres. La valeur de X peut être (). Dans le cas contraire, il s'agit d'une paire pointée:

(nom-de-cellule . attributs)

Le nom de cellule peut être omis. Il est souvent intéressant de ne nommer une cellule qu'à sa fermeture. En particulier, dans les démarches récursives, bien que partant de la cellule la plus grande, on ne connaît sa structure que lorsque les cellules situées à un niveau de récursivité plus bas sont construites. (Exemple: un additionneur 32 bits

construit à partir de deux additionneurs 16 bits, etc...).

Dans le cadre de cette démarche, lorsque plusieurs cellules sont ouvertes, elles sont empilées. Elles ne sont dépilées qu'à la fermeture de la cellule ouverte postérieurement. Ce mécanisme est basé sur le principe des variables dynamiques en lisp. Il n'est donc pas possible de travailler en même temps sur deux cellules ouvertes.

Les attributs sont des couples (clef . valeur). Ils sont affectés au champ "attributs" de la cellule que l'on vient d'ouvrir. Ils ne sont pas évalués. Ils permettent la sauvegarde de variables, et la mémorisation d'un certain nombre d'informations. En particulier, le concepteur peut y mettre des éléments (clef . valeur) qui lui sont propres.

6.2.3 - Fonctions de base

La méthode d'assemblage, une fois une cellule ouverte est relativement simple.

A partir d'un objet cellule, accédé par la fonction:

(accede-fig nom environnement)

il y a création d'une image, sous forme d'un appel:

(gen-appel cellule . obj-nom)

Cette fonction admet pour argument l'objet cellule que l'on souhaite instancier, et elle retourne l'objet appel qui a été construit.

6.2.3.1 - Fonctions liées à l'appel

Sur cet appel, toutes les procédures qui permettent de spécifier les contraintes ou les paramètres d'assemblage sont applicables.

6.2.3.1.1 - Fonction sur les axes d'assemblage

Il y a la direction d'assemblage:

(direction appel angle)

6.2.3.1.2 - Spécification du routage

Il y a le nom des routeurs, et le "net" que l'on souhaite réaliser. Ceci, lorsque le routeur n'est pas le routeur courant, ou lorsqu'il y a plusieurs routages dans le même assemblage (cas du dévoiement sur plusieurs niveaux technologiques).

(routeur appel nom . net)

6.2.3.1.3 - Attitude des cellules

Il y a également les transformées géométriques que l'on souhaite voir appliquées à la cellule:

(rotation appel angle)
(symetrie appel axe)

6.2.3.1.4 - Contraintes de positionnement relatif

Un assemblage étant un positionnement dans le plan de deux cellules, il convient de fixer les positions relatives des

cellules sur les deux axes (cf. 6.3.1, FIGURE A-2). Le concepteur peut donner des contraintes sur le positionnement des cellules, pour cela un des moyens le plus simple est de mettre des points en coïncidence:

(coincide appel couple-points)

D'autres fonctions, variantes plus ou moins importantes des fonctions que l'on vient de voir peuvent exister, comme les contraintes de positionnement absolu.

6.2.3.1.5 - Déclaration de la vue externe d'une cellule

La dernière fonction applicable à un appel est l'application de remontée de la vision hiérarchique de la cellule instanciée. Ceci est fait de façon implicite par l'intermédiaire des déclarations d'interconnexions, dans le cas de la déclaration d'un "net". Mais le plus souvent, il convient de déclarer explicitement ce qui doit être remonté. Les algorithmes de placement, de calcul de la frontière réelle de l'instance ne travaillent qu'avec ces paramètres. Ceci est traduit par les fonctions sur les listes de points de connexion que l'on souhaite voir remonter, ou ne pas remonter, et les listes identiques pour les ombres. Le choix d'une connexion, ou la décision d'interdir une connexion au delà d'un certain niveau est possible:

(remonte appel liste-cn)
(remonte appel liste-ombre)

Pratiquement tous les objets liés à une cellule peuvent être remontés ainsi, la fonction remonte se contentant de prendre les objets et de les mettre dans la A-liste des attributs de

l'appel en fonction de leur typage.

6.2.3.1.6 - Action de pose d'appel

Une fois les contraintes appliquées à l'appel, le concepteur peut appliquer l'action de pose dans la cellule:

(pose cellule appel contexte)

6.2.3.2 - Actions liées à la pose d'un appel

La pose d'un appel dans une cellule d'assemblage déclanche un certain nombre d'opérations au niveau du système.

-Tout d'abord, il convient de vérifier la cohérence des paramètres associés à l'appel, en fonction du contexte d'assemblage. Il y a détection des erreurs de cohérence au niveau des attributs de l'appel, et détection des incompatibilités entre attributs (contraintes opposées par exemple).

-Ensuite, la construction de l'image externe de l'instance est réalisée en fonction de l'attitude choisie, des contraintes appliquées, des connexions déclarées, et des objets remontés.

-Finalement, après interrogation d'un routeur, lorsque cela est demandé, il y a positionnement exact de l'instance, matérialisation des interconnexions, résolution des ombres, et calcul de la nouvelle vision externe de la cellule en fonction du nouvel appel, des interconnexions, et des objets remontés.

6.2.4 - Fonctions lisp complémentaires

Les procédures qui déterminent les paramètres d'assemblage sont d'autant plus puissantes qu'elles peuvent accéder aux élements qui composent la vision externe d'une cellule par des critères de sélection et de filtrage. D'où la fonction d'accès aux objets d'une cellule par les propriétés situées dans son champ "attributs".

6.2.4.1 - Exemple de filtrage

Le choix de l'ensemble des points de connexion situés sur le côté "c-0" d'une cellule "CELLULE", ayant l'attribut "TYPE", associé à la valeur "SORTIE", est traduit par la fonction lisp:

```
(filtre-valeur
      (filtre-attrib
            (filtre-angle (connecteurs CELLULE) 0)
            "TYPE"
      "SORTIE")
```

6.2.4.2 - Principe d'utilisation

Les attributs ont un rôle de sélection et de filtrage dans le cadre d'une programmation orientée objet. Il est alors possible de poser des questions aux objets, et d'étendre de façon dynamique le type d'information mémorisé. Seule la méthode de rattachement à l'objet, et la méthode d'accès aux informations reste invariante (manipuler la structure en dehors des fonctions système est fait au péril de l'utilisateur).

6.2.4.3 - Fonctions d'accès aux attributs

Il n'a qu'une seule fonction de manipulation des attributs d'un objet manipulé (cellule, appel, ombre, connexion):

(attribut OBJET ATTRIBUT . VALEUR)

Tout comme les fonctions CEYX (41) d'accès aux champs d'un objet, cette fonction admet deux ou trois arguments.

Lorsqu'il y a deux arguments, on est en lecture. Le couple (ATTRIBUT . VALEUR) est retourné. Lorsqu'il y a trois arguments, le champ VALEUR est forcé avec la nouvelle valeur.

Si l'argument "ATTRIBUT" n'est pas trouvé; en lecture la valeur () est retournée, en écriture une nouvelle paire pointée (ATTRIBUT . VALEUR) est créée et ajoutée à la liste des attributs de l'OBJET traité.

6.2.4.4 - Exemples d'attributs

-Le nommage: IDENTIFICATEUR
EQUIPOTENTIELLE
DOCUMENTATION

```
(attribut CELLULE   "ID"   OBJ_NOM)
(attribut CONNEXION "EQUI" "VDD_1")
(attribut POINT     "DOC"  "string")
```

-Le typage: Caractérisation d'une cellule, d'une connexion par sa fonction.

```
(attribut CELLULE   "TYPE" "UAL_32")
(attribut CONNEXION "TYPE" "SORTIE")
```

-Les processeurs utilisés lors de la création d'une cellule, et les paramètres associés:

```
(attribut CELLULE "GENERATEUR" '(procédure ...))
(attribut CELLULE "GENERATEUR" '(edit ...))
```

-Les valeurs associées à certaines variables, par exemple la technologie:

```
(attribut CELLULE "TECHNO" 'HCMOS3)
```

A ces fonctions, il convient d'ajouter certaines fonctions d'information hiérarchique, qui font appel aux mécanismes de remontée avec transformée géométrique lorsque nécessaire:

```
(position 'appel 'cn)
(nom      'appel 'cn)
```

Et les fonctions de sauvegarde de la structure de donnée générée:

```
(sauve-fig 'cellule 'contexte . paramètres)
```

Les paramètres sont des valeurs liées à la sauvegarde, et qui permettent de savoir ce qu'il convient de sauvegarder.

Les opérateurs de contraintes par coïncidence de connexion ou de liste de connexions, et que l'on applique à un objet "appel" avant assemblage sont:

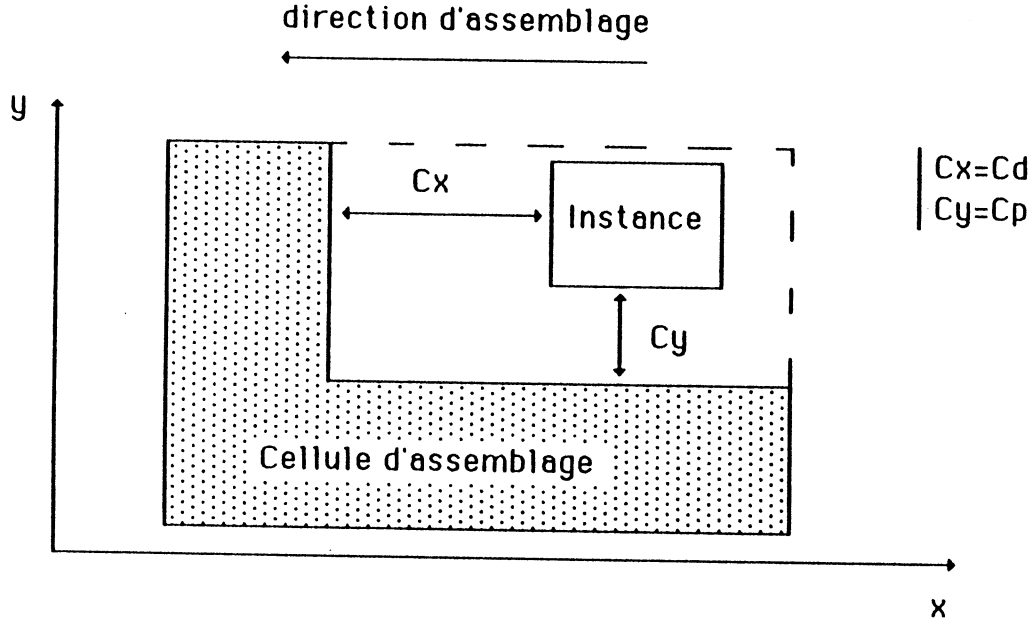


FIGURE A-2

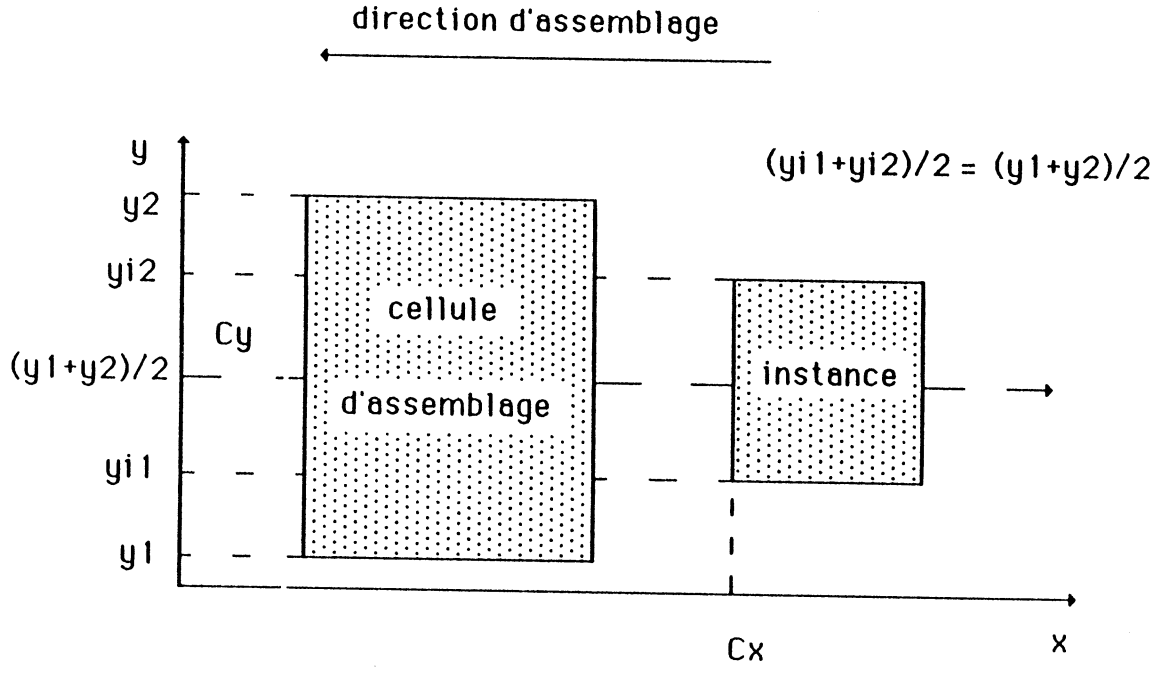


FIGURE A-3

```
(connect 'cellule 'appel 'cn1 'cn2)
(lconnect 'cellule 'appel 'liste-cn1 . 'liste-cn2)
```

6.2.5 - Conclusion

A partir de ce noyau de fonctions de base, il est possible de concevoir n'importe quelle fonction d'assemblage binaire. Ceci dépend exclusivement de la volonté du concepteur. L'immersion en lisp des fonctions, le choix d'une représentation objet des éléments traités,les possibilités de filtrage, typage des objets et la génération dynamique de ceux-ci sont particulièrement orientés vers une conception algorithmique des circuits.

6.3 - Actions associées aux fonctions d'assemblage

A chaque fonction d'assemblage, décrite au chapitre précédant, est associée un certain nombre d'actions. Afin de simplifier au maximum les algorithmes, l'assemblage est conçu de la manière suivante:

6.3.1 - Principe

A chaque nouvelle cellule instanciée dans une cellule d'assemblage, sont affectées deux contraintes Cx et Cy de positionnement en X et en Y (FIGURE A-2). L'exécution de l'assemblage consiste à résoudre ces deux contraintes, sachant que pour un assemblage dans une direction "d", elles sont fonction des contraintes "Cd" dans la direction "d" et "Cp" dans la direction perpendiculaire.

$$Cx=f(Cd,Cp) \qquad Cy=g(Cd,Cp)$$

L'assemblage devient un problème de résolution de Cd et Cp à partir de l'ensemble des spécifications données par le concepteur.

Lorsque celui-ci demande l'exécution de la pose d'un appel dans la cellule d'assemblage, quatre cas se présentent:

-1: Cd et Cp sont fixés (FIGURE A-2). Il n'y a rien à faire. Le positionnement absolu de la cellule est imposé par le concepteur. Il suffit de vérifier qu'il n'y a pas de violation des règles de dessin sur l'ensemble des frontières.

-2: Cp est fixé. Il s'agit du cas classique d'aboutement. Il convient de trouver la garde entre les deux cellules, et de vérifier sur les frontières perpendiculaires à la direction d'assemblage la non violation des règles de dessin.

-3: Cd est fixé. Il convient, en fonction du type de routeur choisi, de trouver la valeur de Cp qui permet un assemblage optimum. Cette configuration ne peut arriver que dans le cas du routage ou du dévoiement. Lorsque c'est le cas, la valeur est fixée arbitrairement, de telle sorte que les deux cellules soient centrées sur l'axe d'assemblage (FIGURE A-3).

-4: Cd et Cp ne sont pas fixées. Il s'agit du cas où la direction d'assemblage est la seule donnée. Il s'agit de fixer Cp, quitte à le faire de manière arbitraire (cf. cas 3). La configuration est alors celle du cas 2.

Si la résolution des contraintes Cp et Cd n'est pas

possible, il y a déclanchement du mécanisme d'erreur avec mémorisation des variables ayant provoqué l'erreur. Ceci est très important pour la mise au point des programmes par le concepteur. L'expérience d'un logiciel comme LUBRICK (146) montre que le concepteur perd beaucoup de temps à chercher les mécanismes d'assemblage déclanchant une erreur de conception lorsqu'il n'a pas le moyen de suivre en pas à pas les actions effectuées, ou lorsque les anomalies ne sont pas enregistrées.

6.3.2 - Actions liées aux fonctions d'assemblage

6.3.2.1 Ouverture d'une cellule d'assemblage

Cette opération, réalisée par la fonction "open-cell" entraîne la sauvegarde de l'environnemnt courant par empilage du contexte courant dans la pile de sauvegarde.

Il y a test afin de savoir si la cellule existe déja. Dans ce cas, elle ne doit pas être ouverte en édition procédurale. Si elle n'est pas ouverte, elle doit être éditable de façon procédurale. Ceci est vérifié par l'envoi des messages "cellule ouverte" et "cellule algorithmique". Si la cellule n'existe pas, ou si elle n'a pas de nom déclaré, elle est construite dans l'environnement courant (bibliothèque, hiérarchie, noeud, technologie, utilisateur,...).
L'ensemble des compteurs et paramètres d'assemblage sont initialisés, les opérations d'assemblage peuvent lui être appliquées.

6.3.2.2 - Construction de l'appel

La construction de l'appel se fait en deux temps, avec tout d'abord la création de l'enregistrement physique qui va correspondre à l'appel, (par la fonction "gen-appel"), puis la génération de l'ensemble des paramètres affectés à l'objet appel avant sa pose dans la cellule d'assemblage.

La création d'un appel, entraîne la création d'un objet sans attitude particulière, auquel est accroché la liste des points de connexion disponibles, les ombres et les frontières de la cellule à assembler.

Les paramètres d'assemblage sont classés en trois catégories, les contraintes topologiques, les connexions réalisées et les objets remontés. Ceci est traduit par l'affectation de valeurs aux couples (clef . valeur) correspondant aux clefs (contrainte, connexions, et remontées) et que l'on place dans la A-liste des attributs de l'appel. Lors de la pose de l'appel, c'est ici que les algorithmes viennent chercher leurs paramètres.

Chaque contrainte se traduit par l'ajout de nouveaux éléments à la liste des attributs de l'appel. Ce sont les algorithmes de pose qui les interprètent.

Les contraintes sont: la direction d'assemblage, toutes les informations d'interconnexion (routeurs, "nets", listes de connexion, de non connexion, les transformées géométriques, les contraintes de placement absolu ou relatif, et les éléments que l'on souhaite voir remonter ou disparaître).

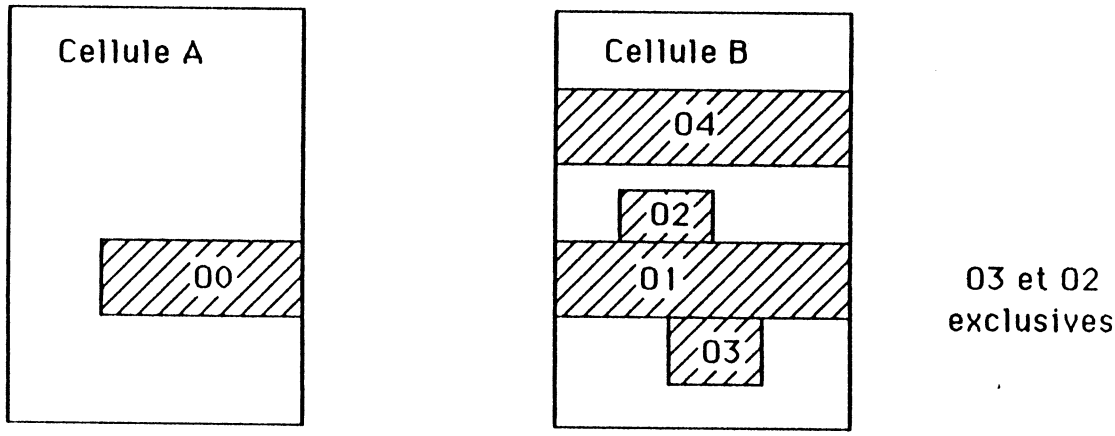
6.3.2.3 - Pose de l'appel

La pose de l'appel suppose que toutes les contraintes ont été fournies. Le travail consiste tout d'abord à vérifier leur cohérence, puis à réaliser la pose de l'appel.

La première action effectuée est la recherche de la direction d'assemblage, celle-ci peut être, soit explicite, soit implicite, ce qui est le cas si le concepteur se contente de spécifier une coïncidence de points de connexion. La direction est donnée par l'orientation des ombres qui doivent coïncider.
A ce niveau, il y a vérification que tous les paramètres pouvant indiquer une direction ne sont pas contradictoires, auquel cas il y a déclanchement d'erreur, avec les valeurs incohérentes.

Une fois la direction d'assemblage fournie, suivant les valeurs des contraintes Cd et Cp, un premier placement approximatif de l'appel est effectué par la juxtaposition des "Bbox" (cf. chapitre frontières). Les frontières sont remontées avec l'attitude de la cellule instanciée (à moins de l'avoir été lors d'un assemblage précédant).
A partir des listes d'interconnexions et des éléments remontés, les listes de points de connexion et d'ombres de la cellule appelée sont remontées avec l'attitude associée, et le calcul de la frontière exacte, dans la direction d'assemblage est effectué. Il se caractérise par la liste des frontières par niveau technologique, et la liste des arêtes où une violation de garde est permise.

A partir de là, l'algorithme de mise à la garde peut être exécuté (cf. frontières). Dans le cas du dévoiement, cette étape est faite en deux parties, avec tout d'abord



Déclaration de 03 ombre de connexion remontée

Assemblage de A et B

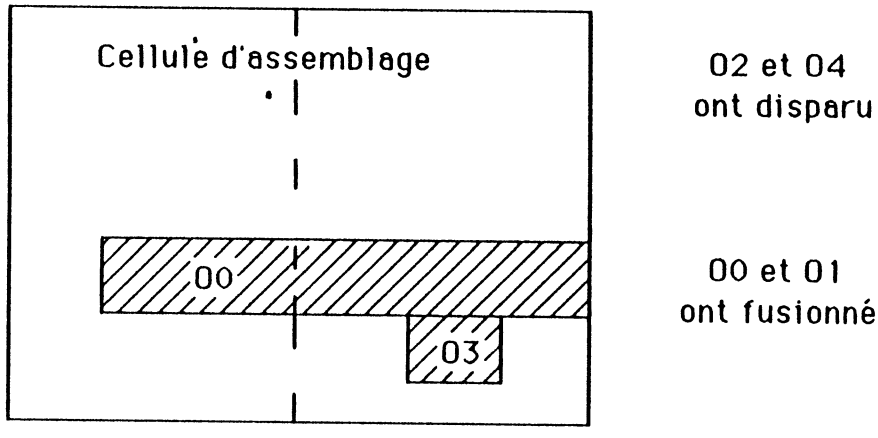


FIGURE A-4

l'exécution du dévoiement avec tassage sur la cellule d'assemblage, calcul de la cellule résultante, puis aboutement de cette cellule avec l'appel. Le mécanisme utilisé est le même pour le routage bi-couches.

Le calcul de mise à garde fait intervenir les arêtes liées à un passage sur la cellule. Lorsque l'algorithme entre dans une zone où il y a passage, le "dmin" est fourni par l'interrogation de l'ombre correspondant à l'arête mise en cause. En réponse, celle-ci fournit la position exacte de l'arête.

Une fois le placement réalisé, toutes les contraintes de positionnement et de connexion doivent être résolues, reste le problème de la remontée des informations hiérarchiques.

6.3.2.4 - Remontée des informations hiérarchiques

Les informations hiérarchiques sont contenues dans les frontières, les ombres et les points de connexion non résolus, et plus généralement dans les champs attributs de l'appel qui vient d'être posé.

A priori, tout ce qui n'est pas résolu est remonté. Cependant, les objets créant un conflit au niveau superieur, ou devenant inaccessibles (cas des connexions n'ayant plus accès au bord de la cellule) sont éliminés.

Pour les frontières, la méthode est simple, la génération de la nouvelle frontière est faite en deux étapes par union de frontières:

frontière=(union

(union (frontière, frontière-du-routage)

frontière-de-l'appel))

Ceci est réalisé avec les algorithmes d'union de frontières et d'ajout d'un élément à une frontière.

Après les frontières, sont remontées les ombres. Celles qui sont liées à des points de connexion qui disparaissent sont éliminées. Celles qui correspondent à une transparence non résolue par composition disparaissent (FIGURE A-4), et celles qui sont en exclusion avec des ombres résolues disparaissent également. Les ombres restantes sont unies avec celles de la cellule d'assemblage, sous forme d'une union de graphes. La vision externe de la cellule d'assemblage est constituée, avec la déclaration des points de connexion de cette nouvelle cellule.

Les attributs qui restent sont affectés aux objets auquels ils étaient destinés. Le pointeur sur la cellule d'assemblage est finalement rendu au concepteur.

Lors de la fermeture d'une cellule, après avoir vérifié la cohérence des données de la cellule, il y a dépilage des cellules d'assemblage ouvertes. Le pointeur sur la cellule d'assemblage est toujours retourné par la fonction de fermeture. Lorsque la pile est vide, l'assemblage est terminé. Le concepteur se doit de donner l'ordre soit de sauvegarde, soit d'édition de la hiérarchie créée, sinon celle-ci est perdue.

6.3.3 - Conclusion sur les actions d'assemblage

Dans un langage d'assemblage, la quantité d'information manipulée, ou entrant en jeu est très importante. Il convient de concevoir des mécanismes permettant d'accéder rapidement à la structure de données, ou permettant un calcul rapide des données, le volume de ces dernières étant considérable. C'est cette approche qui a guidé l'ensemble des choix effectués.

Un autre critère important, est l'indépendance technologique des assembleurs. En offrant la possibilité de spécifier les contraintes de placement par des coïncidences de points, les spécifications, technologie dépendantes, sont transformées en des variables calculées lors de l'assemblage. Le changement d'une technologie (n'entraînant pas de modifications majeures sur les cellules de base et sur leurs interconnexions) ne doit pas être une pénalisation pour le système. A priori, l'algorithme de composition doit pouvoir être réexécuté sans trop de problèmes, avec une réalisation optimale. Ceci est très important pour la réalisation de circuits aux limites de la technologie, pour lesquels le fonctionnement vient influer sur les règles par de petites variations. La regénération des masques du circuit est alors pratiquement instantanée une fois les cellules de base retouchées.

CONCLUSION

L'utilisation d'une représentation symbolique pour la génération des masques de circuits intégrés est, aujourd'hui, admise par tous. La description de circuits sous forme de programmes l'est également.

Un autre point qui apparaît est la volonté de séparation, au niveau de la conception, entre le niveau comportemental, le niveau architectural et le niveau des masques. Il y a recherche d'une indépendance entre ces niveaux, les interfaces étant spécifiés, et les gens qui travaillent sur les différents niveaux étant différents.

La conception de circuits devient structurée, avec une utilisation de générateurs spécialisés, comprenant des optimisations sur le plan topologique ou électrique. L'utilisation de ces générateurs permet l'utilisation de langages de description souples et la paramétrisation des descriptions. Lorsque l'on a besoin d'une réalisation aux limites d'une technologie, on utilise un générateur spécialisé, basé sur des techniques d'assemblage métrique et des cellules adaptées à la technologie employée (76). Les contraintes électriques fines sont maîtrisées. Lorsque l'on a besoin de fonctions; sans grandes contraintes sur les performances, l'utilisation de bibliothèques de cellules paramétrées, plus ou moins indépendantes de la technologie employée, apporte une grande vitesse de conception. Lorsque les contraintes de surface et de performances ne sont pas trop importantes, les langages procéduraux associés à des bibliothèques de cellules paramétrées deviennent particulièrement efficaces.
La dernière contrainte à imposer aux outils, pour obtenir une efficacité maximale est que l'ensemble des outils se

trouve dans une base intégrée. Chaque générateur doit être capable de fournir aux outils d'analyse, de simulation et de vérification, les données nécessaires à leur bon fonctionnement.

Au niveau des langages de génération des masques, et des assembleurs, il semble nécessaire d'assister d'avantage le concepteur, afin de mieux maîtriser les structures construites. Pour cela, le développement d'outils de vérification, comparant la structure réalisée à des modèles prédéfinis, comme on en voit dans la conception assistée de programmes (180), devrait avoir lieu.

Au niveau des systèmes complets de CAO VLSI, ils semblent s'orienter vers l'utilisation de matériel peu coûteux et de grande série. Il y a près de 24 millions de PC/AT, et ces machines supportent des applications de plus en plus complexes en s'intégrant dans la grande informatique, réseaux ethernet, système Xenix, ... De grandes sociétés, comme Daisy Systems Corporation envisagent sérieusement le portage de ses logiciels sur la famille de matériels IBM PC/AT et compatibles (159). Avec cette approche, les systèmes de CAO devraient voir leur prix d'achat s'effondrer, passant de systèmes spécialisés, sur machines lourdes et coûteuses, à des systèmes ouverts, simplifiés, mais tout aussi performants, car livrés à la très dure concurrence des logiciels et réseaux développés sur machines grande série. En ce qui concerne les performances de ces stations à venir, il est déja possible de dire que les cartes graphiques développées sur ce type de machines commencent à devenir très performantes, les cartes spécialisées interface et calcul numérique aussi. Les

mémoires de masse deviennent énormes et à un prix dérisoire. Enfin les logiciels de programmation y sont très performants (Le_Lisp, Pascal, C). La réalisation de coprocesseurs spécialisés va les rendre aussi performantes que des machines spécialisées (exemple: le processeur lisp LLM3-43).

ANNEXE I

Liste non-exhaustive des générateurs de layout

AGS (4) : entrée graphique des masques

ALI (101) : textuel et masques + graphe électrique, immergé en PASCAL, structuration hiérarchique de pascal

ALLENDE (117) : assembleur de cellules, immergé PASCAL et C, suit ALI.

ALPHA (2) : dessin top-down, routage global à partir cell prédéfinies.

AMOEBA (104) : système symbolique interactif, graphe électrique.

ASTRA (133) : système CAO, editeur symbolique, assembleur, generateur de cellules.

BASYLIC (10) : dessin interactif de cellules avec "DRC" et représentation métrique.

BBL (44) : Berkeley building block layout system.

BRISTLE BLOCKS (81) : textuel d'assemblage.

CADDS II (36) : graphique + textuel des masques.

Caesar (124) : entrée graphique des masques.

CAF (98) : plan de masse interactif.

CAMELEON (153) : éditeur graphique LUDIEC, symbolique et compactage, basé sur le langage TDL de description des symboles, SPACER utilisé pour placer les objets à la garde, PLASCO générateur de PLA, ROM. Environnement de générateur de modules en lisp.

CASSIOPE (42) : environnement de CAO, éditeur logique, mdmos, stick, électrique, base de donnée LOF.

CDS (65) : 1970s, suit ICARUS, cell design system, graphique interactif des masques.

CHIPMASON/PLAYOUT (94) : programmes de génération de structures régulières.

Chipmunk () : graphique des masques.

CHISEL (84) : extension C, concept de cellule avec des ports, routage entre blocs.

CIF (156) : description textuelle des masques.

DPL/Daedalus (16) : textuel (procédure lisp) + graphique des masques, base de données orientée objet.

DRAWSYM (6) : générateur de layout incorporant paramètres électriques,(éléments parasites), et symbolique.

DUMBO (162) : compilation du modèle de graphe en un symbolique compacté.

Earl/SLAP (89) : assembleur stretch + contraintes, textuel + graphe électrique. EARL interprété, SLAP immergé en SIMULA.

ELECTRIC (143) : textuel + graphique + graphe électrique.

ESCHER (51) : conception récursive incrémentale.

FLOSS (48) : symbolique + graphe de contraintes (1976), compactage d'assemblage ...

FRED (189) : base de données procédurale pour l'architecture et le plan de masse. Immergé en lisp, utilise de générateurs de modules.

GENERIC (155) : langage utilisé par des compilateurs, fait appel à des générateurs, représentation symbolique relatif + contraintes

HCL (29) : structure de donnée pour symbolique virtuel relatif et structuration hiérarchique, contraintes sur les connexions.

HDL (18) : support des versions d'un layout.

Hed and Fatfreddy (15) : éditeur hiérachique et symbolique.

HIDELA (130) : langage de description d'interconnexions et de placement relatif. Description de la cellule par contraintes, forme logique.

HILL (99) : environnement de dessin hiérarchique stick, langage hiérarchique, compacteur, simulateur switch, immergé PASCAL.

HSL-FX (73) : langage unifié de conception (fonctionnel-masques).

I () : textuel + graphe électrique.

ICARUS (62) : graphique interactif des masques.

ICLIC () : textuel des masques.

ICPL (93) : version "hp" de DPL.

IGS (75) : éditeur incrémental + "DRC" incrémental.

KBES (91) : système expert de génération.

LAP (103) : cellule, textuel immergé en SIMULA, génère du CIF.

LAVA () : textuel + graphique + graphe électrique, composition de cellules.

LAYLA (53) : langage PASCAL de génération de cellules.

LDMC () : langage C de construction, à partir de LDM,

langage géométrique.

LOF (20) : langage où les cellules sont construites par assemblage, outil de construction de librairies de blocs flexibles.

LUBRICK (146) : assembleur de cellules immergé en PASCAL.

LUCIE (128) : textuel + graphique des masques.

Lucien () : langage de description de masques.

LUCIFER (43) : langage lisp de génération de masques.

MAC PITTS (152) : compilateur type microprocesseur.

MADMACS (66) : éditeur graphique hiérarchique de masques.

Magic (125) : système, structure de donnée hiérachique, extracteur, "DRC", "stretch" et compactage, (PICTURES with PARENTHESES utilisation de procédures de construction).

MGE (154) : environnement de création de générateurs de modules (cf. PLASCO).

MGX (127) : éditeur "stick", éditeur masques, vérificateur de connectivité, extracteur de paramètres électriques.

MIMOLA (177) : GEMS éditeur graphique, génération possible à partir de PASCAL, FORTRAN.

MOVE (21) : cadre du projet SILVER, manipule le positionnement relatif des cellules.

MULGA (181) : textuel + graphique des masques + graphe électrique (ICDL langage de description).

NS (45) : symbolique grille virtuelle, environnement lisp, langage type DPL, éditeur graphique, générateur de parties opératives.

PAL () : langage textuel des masques.

PALLADIO (30) : descriptif de haut niveau+graphique, experimental.

PAUL : cellule, proche de LAP, format de sortie "stick".

PHLED (95) : une version de CAESAR supportant le 45°.

PI (134) : système d'interconnexion en lisp.

PLATES (144) : textuel, symbolique non métrique, positions relatives, compactage.

Pooh (183) : Représentation pour entrée graphique, textuelle, programmes.

PSI (64) : similaire à STICKS, CABBAGE, REST. Correct par construction, éditeur interactif, SPACER (compacteur), "maskmaker" (génère les masques). Représentation textuelle.

PUDIC (173)' : procédures PASCAL de description des cellules, sortie CIF.

REST (119) : cellule, graphique + graphe électrique.

RIOT (167) : outil graphique d'interconnexions de VTI.

RPL (135) : langage symbolique sur les rectangles avec contraintes de placement.

SAGA (161) : assembleur experimental, génère du CIF. Langage d'entrée SIL. Langage Algol-like.

SAM (165) : cellule, écrit en SMALLTALK, graphique +texte.

SCALD (85) : management des informations de dessin par assemblage. Approche intégrée.

SCALE (109) : génération automatique, symbolique + masque, (cellule spéciale), séparation cellule/hiérarchie, génère la forme intermédiaire IDL.

SDL (34) : ensemble d'outils de dessin intégré, union graphique électrique, conception par procédures. Le langage utilisé est L, langage proche de C ayant son propre parseur.

SELLAV (26) : dessin interactif ou génération par procédures, cellules symboliques et composition, simulateur, compacteur, placement routage.

SHARPS (46) : outil de placement routage, textuel.

SIDS (52) : éditeur symbolique + "DRC" incrémental.

SILCO (130) : générateur d'assemblage par compilation d'un langage paramétré.

SILT (50) : textuel + graphique des masques.

SILVAR - LISCO (175) : station de travail, langage procédural avec éditeur incrémental, compacteur hiérarchique.

SKILL (97) : langage ayant une syntaxe PASCAL, immergé en lisp de génération procédurale. Dessin par modules générateurs.

SLIM (60) : compacteur global par graphe de contraintes.

SLIP (47) : symbolique + compaction.

SPACER II (59) : Compacteur.

SPAM () : description hiérarchique haut niveau.

SPARCS (35) : grille relative, virtuelle, symbolique avec contraintes. Travaille sur la base d'objets OCT. Création du graphe des contraintes et résolution.

SPHINX IC (188) : base de donnée d'édition de masques et vérification, description électrique par BDL, représentation hiérarchique.

SPRINT (179) : conception hiérarchique, cellule - comportementale, structurelle, électrique. Deuxième hiérarchie - plan de masse, éditeur de plan de masse, langage HCD de description, tout intégré, dessin de 100 mille transistors.

SQUID (87) : base de données avec masques, symbolique, simulation et schémas.

STICKS (186) : symbolique + compaction.

Stick and Stones (40): textuel + graphe électrique.

SYMCAS () : environnement interactif symbolique pour le dessin de modules. Elements graphiques ALPs. Symbolique hiérarchisé. Relations de proximité entre élements.

SYMPLE (160) : outil de layout symbolique pour bipolaire.

TOPOLOGIZER (90) : système expert de dessin de cellules.

UCAD (164) : outils sous UNIX, éditeur graphique et langage SGL.

VANGUARD (71) : système de dessin combinant "gate-array" et "full-custom", partiton du circuit.

VILE (7) : éditeur interactif, sortie CIF, niveau des masques.

VIVID (139) : assemblage, symbolique virtuel (ABCD langage de description).

VTI (168) : Station intégrée, masques, électrique, logique.

REFERENCES

(1) B.Ackland, N.Weste,
An automatic assembly tool for virtual grid symbolic layout,
VLSI'83, F.Anceau and E.J.Aas Ed., Elsevier Science
Publishers B.V.(North-Holland), IFIP, 1983, pp.457-456.

(2) T.Adachi, H.Kitazawa, M.Nagatani, T.Sudo,
Hierarchical Top-Down layout design method for VLSI chip,
19th Design Automation Conference, 1982, pp.785-791.

(3) M.Adiba, C.Delobel,
Bases de données et systèmes relationnels,
Dunod, Paris, 1982.

(4) AGS/860 V3.0 User's guide,
Applicon Inc., Burlington, Mass., September 1981.

(5) S.B.Akers, J.M. Geyer, D.L.Roberts,
IC Mask Layout with a Single Conductor Layer,
7th Design Automation Conference, June 1970, pp.7-16.

(6) D.R.Alessandrini,
DRAWSYM: a transistor or gate level symbolic layout system,
ICCAD'84, 1984, pp.281-283.

(7) G.H.Allen, S.Perry,
A VLSI Interactive Layout Editor (VILE),
Software-Practice and Experience, 1985, v.15(8), pp.795-806.

(8) F.Anceau, R.Reis, IMAG France,
Complex integrated circuit design strategy,
IEEE Journal of solid state circuits, vol.SC.17, n.3, pp.459-
464, June 1982.

(9) F.Anceau,
Génération versus immersion et paramétrisation,
Note Sycomore, 19 Mai 1984.

(9-1) F.Anceau,
STYX: Descriptions squelettisées normées,
Projet Sycomore, note IMAG, Grenoble, 20 Juin 1983.

(10) A.M.Anckaert, C.Trullemans,
BASYLIC: an interactive cell-design tool,
ESSIRC 84, 1984, pp.123-126.

(11) R.F.Ayes,
IC Design under ICL, Version 1,
Caltech SSP Report 1366 (revised 4031),
California Institute of Technology, 1978.

(12) M.W.Bales,
Layout Rule Spacing of Symbolic Integrated Circuit Artwork,
ERL Memo n.UCB/ERL M82/72, Berkeley, May 1982.

(13) F.Bancilhon, W.Kim, H.F.Korth,
Transactions and concurrency control in CAD data bases,
ICCD-85, Rye town, pp.86-89, October 1985.

(14) M.R.Barbacci,
ISPS: Instruction Set Processor Specifications, the notation
and its application,
IEEE Transactions on Computers C-30(1):24-40, January, 1981.

(15) E.Barton,
Graphic Design Aids: HED and FATFREDDY,
VLSI Architecture, B.Randell and P.C.Trealeaven ed.,
Prentice Hall International, 1983.

(16) J.Batali and A.Hartheimer,
The Design Procedure Language Manual, AI Memo 598,
Massachusetts Institute of Technology, September 1980.

(17) J.Batali, N.Mayle, H.Shrobe, G.Sussman, D.Weise,
The DPL/Daedlus Design Environment,
VLSI'81, 1981, pp.183-192.

(18) D.S.Battory, W.Kim,
Modeling concepts for VLSI CAD objects,
ACM transactions on data base systems, v.10, n.3, 1985.

(19) D.S.Battory, W.Kim,
Support for versions of VLSI CAD objects, TR-85-18,
Department of Computer Science, University of Texas, Austin,
September 1985.

(20) J.M.Berge, L.O.Donzelle, V.Olive, J.Rouillard,
D.Rouquier,
Development and use of flexible block libraries,
ESSCIRC'86, 1986, pp.28-30.

(21) N.Bergmann,
MOVE - A Useful Primitive for a Variety of I.C. C.A.D.
Tools,
Department of Computer Science, University of Edinburgh.

(22) A.M.Beyls, B.Hennion, J.Lecourvoisier, G.Mazare,
A.Puissochet,
A design methodology based upon symbolic layout and
integrated CAD tools,
19th Design Automation Conference, 1982, pp.872-878.

(23) G.Birtwistle, O.J.Dahl, B.Myhrhaug, K.Nygaard,
SIMULA,
Studentlitteratur, Lund, Sweden, 2nd ed., 1979.

(24) D.G.Bobrow, T.Wingrad,
An overview of KRL, a knowledges representation language,
Cognitive Science, n.1, v.1, 1977.

(25) D.G.Bobrow, M.Stefik,
The LOOPS manual,
Xerox memo, VLSI-81-13, 1982.

(26) W.Bonath, M.Glesner, G.Schaefer, N.Wehn,
SELLAV: A New Symbolic Design Method and its System
Implementation,
VLSI'85, E.Hörbst ed., Elsevier Science Publishers B.V.,
North-Holland, IFIP 1986, pp.293.

(27) W.Bonath, M.Glesner, G.Schaefer, N.Wehn,
A Knowledge Base Approach in a Symbolic Design System,
Technical University Darmstadt, Germany.

(28) E.Bourcier,
Conception et Realisation du simulateur Irene-C,
CUEFA, Mémoire d'ingénieur CNAM, Grenoble, Oct. 1984.

(29) F.D.Brewer,
HCL-A System For Hierarchical Layout, thesis, report n.UIUCDCS-R-84-1194,
University of Illinois, Urbana-Champain, December 1984.

(30) H.Brown, C.Tong, G.Foyster,
PALLADIO: An Exploratory Environnement of Circuit Design,
IEEE Computer, December 1983, pp.41-56.

(31) R.Bryan,
A Switch Level Simulation Model for Integrated Circuits,
MIT Laboratory for Computer Science Technical Report, pp.259

(32) I.Buchanan,
Modelling and Verification in Structured Integrated Circuit Design, PhD Thesis, 1980,
Department of Computer Science, University of Edinburgh.

(33) I.Buchanan,
SCALE:A VLSI Design Langage, technical report CSR-117-82,
University of Edinburg, Department of Computer Science, May 1982.

(34) M.R.Burich,
Design of module generators and silicon compilers,
Nato Advanced Study Institute on Logic Synthesis and Silicon Compilation for VLSI Design, SSGRR - l'Aquila (Italy), July 7-18, 1986.

(35) J.L.Burns, A.R.Newton,
SPARCS: A New Constraint-Based IC Symbolic Layout Spacer,
IEEE 1986 Custom Integrated Circuits Conference, pp.534-539.

(36) CADDS 2/VLSI Product Specifications,
Computervision Corporation,
Bedford, Massachusetts, February 1982.

(37) J.P.Caisso,
Simulation multivaluée, simulation switch,
Rapport de Recherche 597, TIM3-IMAG, Grenoble, Mars 1986.

(38) CALMA - GDS2,
Calma Company,
2901 Tasman Drive, Santa Clara, California 95050.

(39) L.Cardelli,
The Sticks and Stones Painter's Manual,
Department of Computer Science, University of Edinburgh,
internal report, 1980.

(40) L.Cardelli,
Sticks and Stones: An Applicative VLSI Design Language,
Department of Computer Science, University of Edinburgh,
internal report CSR-85-81, July 1981.

(41) L.Cardelli, G.Plotkin,
An Algebraic Approach to VLSI Design,
VLSI'81, J.Gray editor, Academic Press, New York, 1981,
pp.173-182.

(42) Cassiopée,
Manuel d'utilisation,
Département RCA, CNET Grenoble C.N.S., Septembre 1984.

(43) J.Chailloux, J.M.Hullot, J.J.Levy, J.Vuillemin,
Le système Lucifer d'aide à la conception de circuits intégrés,
Rapport de recherche INRIA, n.196, Mars 1983.

(44) N.P.Chen, C.P.Hsu, E.S.Kuh,
The Berkeley Building-Block (BBL) Layout system for VLSI design,
VLSI'83, F.Anceau and E.J.Aas Ed., Elsevier Science Publishers B.V.(North-Holland), IFIP, 1983, pp.37-44.

(45) J.Cherry, H.Shrobe, N.Mayle, C.Baker, H.Minsky, K.Reti, N.Weste,
NS: an Integrated Symbolic Design System,
VLSI'85, E.Hörbst ed., Elsevier Science Publishers B.V., North-Holland, IFIP 1986, pp.325-342.

(46) T.Chiba, N.Okuda, T.Kambe, I.Nishioka, T.Inufushi, S.Kimura,
SHARPS: a hierarchical layout system for VLSI,
18th Design Automation Conference, 1981, pp.820-827.

(47) Y.E.Cho,
A Subjective Review of Compaction,
22nd Design Automation Conference, 1985, pp.396-404.

(48) Y.E.Cho, A.J.Korenjak,D.E.Stockton,
FLOSS: an approach to automated layout for high-volume designs,
14th. Design Automation Conference, 1977, pp.138-141.

(49) S.H.Chuquillanqui Bernaola,
Une nouvelle approche pour l'optimisation topologique et l'automatisation du dessin des masques de P.L.A. complexes, thèse,
Institut National Polytechnique Grenoble, Octobre 1984.

(50) J.Clark and T.Davis,
SILT: A VLSI Design Language, technical report,
Stanford University, 1982.

(51) E.Clarke, Y.Feng,
Escher--A geometrical Layout System For Recursively Defined Circuits,
23rd. Design Automation Conference, 1986, pp.650-653.

(52) D.Clary, R.Kirk, S.Sapiro,
SIDS: A Symbolic Interactive Design System,
17th Design Automation Conference, 1980, pp.292-295.

(53) W.E.Cory,
LYLA: A VLSI Layout Language,
22nd Design Automation Conference, 1985, pp.245-251.

(54) M.Cosnard, A.Guyot, B.Hochet, J.M.Muller, H.Ouaouicha, E.Zysman,
FELIN: An elementary function cruncher,
Computers and Computing, P.Chenin, C.DiCrescenzo et F.Robert ed., Masson et Wiley, Janvier 1986.

(55) R.A.Da Luz Reis,
TESS-Evaluateur topologique prédictif pour la génération automatique des plans de masse de circuits VLSI, thèse,
Institut National Polytechnique Grenoble, Janvier 1983.

(56) S.Dami,
La notion d'objet en intelligence artificielle, DEA,
INPG, Grenoble, 3 Juillet 1986.

(57) I.C.Dennison,
A Sticks Front End for Scale,
Department of Computer Science, University of Edinburgh,
internal report CSR-163-84, September 1984.

(58) Digital systems design automation,
Language, simulation and data bases,
Computer Science Press, M.Breuer Ed., 1975.

(59) J.Do, W.M.Dawson,
SPACER II: a well-behaved IC layout compactor,
VLSI'85, E.Hörbst ed., Elsevier Science Publishers B.V., North-Holland, IFIP 1986, pp.283-292.

(60) A.E.Dunlop,
SLIM-The translation of symbolic layouts into mask data,
17th Design Automation Conference, 1980, pp.595-602.

(61) EDIF,
Electronic Design Interchange Format, Version 1 0 0,
Electronic Design Interchange Format Steering Committee,
1985.

(62) D.G.Fairbairn,
ICARUS: an interactive integrated circuit layout program,
15th Design Automation Conference, 1978, pp.188-192.

(63) J.Ferber,
MERING: un langage d'acteurs pour la représentation et la manipulation des connaissances,
Thèse Docteur-Ingénieur, Paris VI, 20 Décembre 1983.

(64) R.D.Fiebrich, Yuh-Zen Liao, G.Koppelman, E.Adams,
PSI: A symbolic layout system,
IBM J. Res. Develop., v.28, n.5, September 1984.

(65) D.Franco, L.Reed,
The Cell Design System,
18th Design Automation Conference, 1981, pp.240-247.

(66) P.Frison, E.Gautrin,
MADMACS: a new VLSI layout macro editor,
23rd Design Automation Conference, 1986, pp.654-658.

(67) D.Gibson, S.Nance,
SLIC: - Symbolic Layout of Integrated Circuits,
13rd Design Automation Conference, June 1976, pp.434-440.

(68) A. Goldberg, D.Robson, Xerox Palo Alto Reasearch Center,
Smalltalk-80, the language and its implementation,
Addison-Wesley Publishing Company, 1983.

(69) M.Gordon,
A Model for Register Transfer Systems with Applicatons to Microcode and VLSI Correctness, technical report CSR-82-81,
Department of Computer Science, University of Edinburgh,
March 1981.

(69-1) I.Guiguet,
Liaison d'un outil de description des circuits intégrés à un microscope électronique à balayage,
Mémoire CNAM, C.U.E.F.A. Grenoble, Octobre 1985.

(70) A.Guyot, B.Hochet, C.Mauras, J.M.Muller, Y.Robert,
SCALA, une cellule systolique programmable pour l'algèbre linéaire et le traitement du signal,
Colloque C3, Angoulême, Juin 1987.

(71) P.S.Hauge, E.J.Yoffa,
VANGUARD: A Chip Physical Design System,
23rd Design Automation Conference, 1986, pp.440-446.

(72) R.W.Hon, C.H.Sequin,
A Guide to LSI Implementation, technical report SSL-79-7,
Xerox Palo Alto Research Center, January 1980.

(73) T.Hoshino, O.Karatsu, T.Nakashima,
HSL-FX: a Unified Language for VLSI Design,
Computer Hardware Description Languages and their Applications, C.J.Koomen and T.Moto-oka eds., Elsevier Science Publishers B.V., North-Holland, IFIP 1985.

(74) M.Y.Hsueh, D.O.Pederson,
Computer-Aided Layout of LSI Circuit Building-Blocks,
IEEE International Symposium on Circuits and Systems, pp.474-477.

(75) B.Infante, D.Bracken, B.McCalla, S.Yamakoshi, E.Cohen,
An Interactive Graphics System for the Design of Integrated Circuits,
15th Design Automation Conference, 1978, pp.182-187.

(76) R.Jamier,
Génération automatique de parties opératives de circuits VLSI de type microprocesseur, thèse,
Institut National Polytechnique Grenoble, Novembre 1986.

(77) R.Jamier, A.A.Jerraya,
APOLLON: a datapath compiler,
ICCD'85, 1985.

(78) A.A.Jerraya, P.Varinot, R.Jamier, B.Courtois,
Principles of the SYCO Compiler,
23rd Design Automation Conference, pp.715-721.

(79) A.A.Jerraya, E.Rosier, F.R.Rougeaux, B.Courtois,
A Hierarchical Symbolic Design Layout Tool: STYX,
VLSI'85, E.Hörbst ed., Elsevier Science Publishers B.V.,
North-Holland, IFIP 1986, pp.335-347.

(80) R.Joobbani,
An Artificial Intelligence Approach to VLSI Routing,
Kluwer Academic Publishers, Boston, Lancaster, 1986.

(81) D.Johannsen,
Bristle Blocks: A Silicon Compiler,
Caltech Conference on VLSI, January 1979, pp.303-310.

(82) D.Johannsen,
Bristle Blocks, A Silicon Compiler,
Proc. of the 16th. Design Automation Conference, 1979.

(83) S.C.Johnson,
Hierarchical Design Validation Based on Rectangles,
Conference on Advanced Research in VLSI, MIT, 1982, 97-100.

(84) K.Karplus,
CHISEL: an Extension to the Programming Language C for VLSI Layout, thesis, report N.STAN-CS-82-959,
Department of Computer Science, Stanford University,
February 1983.

(85) R.H.Katz
A data base approach for managing VLSI design data,
19th Design Automation Conference, Las Vegas, pp.274-282,
Juin 1982.

(86) R.H.Katz, S.Weiss,
Chip Assemblers: Concepts and Capabilities,
20th Design Automation Conference, Juin 1983, pp.25-30.

(87) K.H.Keller, A.R.Newton, S.Ellis,
A Symbolic Design System for Integrated Circuits,
19th Design Automation Conference, 1982, pp.460-466.

(88) C.Kingsley,
A Hierarchical, Error-Tolerant Compactor,
21st Design Automation Conference, 1984, pp.126-132.

(89) C.Kingsley, G.Williams,
Earl - Interpreted circuit design system, technical report,
California Institute of Technology, 1981.

(90) P.W.Kollaritsch, N.H.E.Weste,
TOPOLOGIZER: An Expert System Translator of Transistor Connectivity to Symbolic Cell Layout,
ESSCIRC'84, 1984, pp.175-178.

(91) T.J.Kowalski, D.E.Thomas,
The VLSI Design Automation Assistant: What's in a Knowledge Base,
22nd Design Automation Conference, 1985, pp.252-258.

(92) M.S.Krishnan,
A Structured Approach to VLSI Layout Design,
Caltech Conference on VLSI, January 1981, pp.413-432.

(93) A.J.Kuchinsky, W.A.Barrett,R.G.Rogers, M.J.Sorens, J.S.Gibson,
ICPL: Integrated Circuit Procedural Language,
CH2080-0, IEEE, 1984, pp.802-806.

(94) Kuo-Hsiung Wu, A.C.Parker, K.Conner,
Procedural Layout: Some Practical Experience For Production-Quality Integrated Circuits,
VLSI'83, F.Anceau and E.J.Aas Ed., Elsevier Science Publishers B.V.(North-Holland), IFIP, 1983, pp.447-456.

(95) A.R.Lanfri,
PHLED45: an Enhanced Version of Caesar Supporting 45° Geometries,
21st Design Automation Conference, 1984, pp.558-564.

(96) R.P.Larsen, J.A.Luisi, A.K.Singh,
Interactive Symbolic Design for VLSI Modules,
19th Design Automation Conference, 1982, pp.291-299.

(97) H.S.Law, G.Wood,
A mixed-media approach to module generator design,
Nato Advanced Study Institute on Logic Synthesis and Silicon Compilation for VLSI Design, SSGRR - l'Aquila (Italy), July 7-18, 1986.

(98) A.Leblond,
CAF: a computer-assisted floorplanning tool,
20th Design Automation Conference, 1983, pp.747-753.

(99) T.Lengauer, K.Mehlhorn,
The HILL System: A Design Environment for the Hierarchical Specification, Compaction, and Simulation of Integrated Circuit Layouts,
Conference on Advenced Research in VLSI, M.I.T., January, 1984, pp.139-149.

(100) B.M.Liblong
SHIFT, a Structured Hierarchical Intermediate Form for VLSI Design Tools, thesis,
Department of Computer Science, Calgary, Alberta, 1984.

(101) R.J.Lipton, S.C.Sedgewick, R.Valdes, G.Vijayan,
ALI: a Procedural Language to Describe VLSI Layouts,
Proceedings, ACM IEEE 19th DAC, 1982, pp.467-474.

(102) R.J.Lipton, S.Sedgewick, R.Valdes, S.North, G.Vijayan,
VLSI Layout as Programming,
ACM Transactions on Programming Languages and Systems, v.5, n.3, July, 1983.

(103) B.Locanthi,
LAP: A Simula Package for IC Layout,
Caltech Technical Report Display File 1862, July 1978.

(104) M.Lotvin, B.Juran, R.Goldin,
AMOEBA: a symbolic VLSI layout system,
21st Design Automation Conference, 1984, pp.294-300.

(105) C.Lursinsap, D.Gajski,
Methods for Cell Compilation with Constraints,
CH2223-6, IEEE, 1985, pp.303-307.

(106) J.MacCarthy,
Lisp 1.5 Programmer's Manual,
MIT Press, Cambridge, Massachusetts, 1962.

(107) M.Marcoux, M.Pomiam,
PLASMA: un language pour pour l'intelligence artificielle,
AFCET, IAFC, 1979.

(108) S.Marine,
IRENE: un langage pour la description, simulation et
synthèse automatique du matériel VLSI,
Thèse INPG, IMAG/TIM3, Grenoble, 3 Février 1986.

(109) R.M.Marshall, I.Buchanan,
SCALE, A Language for VLSI Design, internal report,
Department of Computer Science, University of Edinburgh,
Scotland, 9 January 1984.

(110) A.Martinez, S.Nance,
Methodology for Compiler Generated Silicon Structures,
21st Design Automation Conference, 1984, pp.689-691.

(111) R.Matthews, J.Newkirk, P.Eichenberger,
A Target Language for Silicon Compilers,
24th IEEE Computer Society Int. Conf., 1982, 349-353.

(112) R.N.Mayo, J.K.Ousterhout,
Pictures with Parentheses: Combining Graphics and Procedures
in a VLSI Layout Tool,
20th Design Automation Conference, 1983, pp.270-276.

(113) R.C.McGarity, D.P.Siewiorek,
Experiments with the SLIM Circuit Compactor,
20th Design Automation Conference, 1983, pp.740-746.

(114) C. Mead, L. Conway,
Introduction to VLSI systems,
Addison-Wesley Publishing Company, 1980.

(115) M.Minsky,
A Framework for Representing Knowledge,
Psychology of Computer Vision, P.H.WINSTON Ed., pp 211-227,
Mc Graw-Mill, 1975.

(116) L.Monier, J.Vuillemin,
Using Silicon Assemblers,
VLSI'85, E.Hörbst ed., Elsevier Science Publishers B.V.,
North-Holland, IFIP 1986, pp.325-342.

(117) J.Monteiro da Mata,
ALLENDE: a Procedural Language for the Hierarchical
Specification of VLSI Layouts,
22th Design Automation Conference, 1985, pp.183-189.

(118) J.Monteiro da Mata,
A methodology for VLSI design and, a constraint-based layout language, thesis,
Department of Electrical Engineering and Computer Science,
Princeton University, October 1984.

(119) R.C.Mosteller,
REST: A Leaf Cell Design System, M.Sc. Thesis,
Caltech, Silicon Structures Project Technical Report 4317,
December 1981.

(120) R.C.Mosteller,
REST: A Leaf Cell Design System,
VLSI'81, J.Gray editor, Academic Press, New York, 1981,
pp.163-172.

(121) L.W.Nagel,
SPICE2: A Computer Program to Simulate Semiconductor Circuits, ERL Memo ERL-M520,
University of California, Berkeley, May 1975.

(122) A.R.Newton,
Symbolic Layout and Procedural Design,
NATO Advanced Study Institute on Logic Synthesis and Silicon Compilation for VLSI Design, SSGRR - l'Aquila (Italy), 1986.

(123) R.H.J.M. Otten, IBM Yorktown Heights, New York,
Layout Compilation,
NATO Advanced Study Institute on Logic Synthesis and Silicon Compilation for VLSI Design, SSGRR - l'Aquila (Italy), 1986.

(124) J.K.Ousterhout,
Caesar: an Interactive Editor for VLSI Layouts,
VLSI Design II-4, 1981, 34-38.

(125) J.K.Ousterhout,
Corner Stitching: A Data-Structuring Technique for VLSI Layout tools,
IEEE transactions on computer-aided design, v. CAD-3, n.1,
January 1984.

(126) J.K.Ousterhout, G.T.Hamachi, R.N.Mayo, W.S.Scott, G.S.Taylor,
MAGIC: A VLSI Layout System,
21st Design Automation Conference, 1984, pp.152-159.

(127) M.Ozaki, M.Watanabe, M.Kakinuma, M.Ikeda, K.Sato,
MGX: an Integrated Symbolic Layout System for VLSI,
21st Design Automation Conference, 1984, pp.572-579.

(128) J.F.Paillotin,
Le système Lucie, version du 1er Juillet 1985,
Manuel, TIM3-IMAG, INP Grenoble.

(129) T. Perez Segovia,
Paola: Un système d'optimisation topologique de P.L.A.,
Thèse de Docteur 3eme cycle INPG, 1985.

(130) C.Piguet,
Eléments d'un compilateur de silicium,
Colloque des Logiciens d'Expression Française, CLEF II,
Louvain la Neuve, 11-12 Septembre 1984.

(131) Precision Artwork Language (PAL),
Automation Technology Inc., 1971.

(132) F.Rechenman, A.Bensaid, D.Garnier,
SHIRKA: des systèmes experts centrés objets,
Les systèmes experts, leurs applications, Avignon, Mai 1985.

(133) M.C.Revett, P.A.Ivey,
ASTRA: a CAD system to support a structured approch to IC design,
VLSI'83, F.Anceau and E.J.Aas Ed., Elsevier Science Publishers B.V.(North-Holland), IFIP, 1983, pp.413-422.

(134) R.L.Rivest,
The "PI" (placement and interconnect) System,
19th Design Automation Conference, 1982, pp.475-481.

(135) J.A.Roach,
The Rectangle Placement Language,
21st Design Automation Conference, 1984, pp.405-410.

(136) C.Roche,
EAQUE-LRO: Génération de systèmes experts. Application au problème de l'ordonnancement,
Thèse 3eme cycle, INPG, Grenoble, 4 Juillet 1984.

(137) C.Roche,
LRO2,
2emes journées d'études sur les langages orientés objets,
Brest, Novembre 1984.

(138) C.Roche, J.P.Laurent, P.Clement,
Description du langage LRO2, rapport technique, Projet SMP, contrat ADI 85000530, août 1985.

(139) C.D.Rogers, J.B.Rosenberg, S.W.Daniel,
MCNC's Vertically Integrated Symbolic Design System,
22nd Design Automation Conference, 1985, pp.62-68.

(140) J.B.Rosenberg,
Chip Assembly Techniques for Custom IC Design in a Symbolic Virtual-Grid Environment,
Conference on Advanced Research in VLSI, M.I.T., 1984, pp.213-225.

(141) J.B.Rosenberg, D.Boyer, J.Dallen, S.W.Daniel, C.Poirier, J.Poulton, C.D.Rogers, N.Weste,
A Vertically Integrated VLSI Design Environment,
20th Design Automation Conference, 1983, pp.31-38.

(142) J.A.Rowson,
Understanding Hierarchical Design, Ph.D. thesis,
Caltech Technical Report 3710, April 1980.

(143) S.M.Rubin,
An Integrated Aid for Top-Down Electrical Design,
VLSI'83,F.Anceau and E.J.Aas Editors,
Elsevier Science Publications B.V. (North Holland)
IFIP, 1983.

(144) S.Sastry, S.Klein,
PLATES: a metric-free VLSI layout language,
Conference on Advanced Research in VLSI, M.I.T., 1982,
pp.165-174.

(145) J.P.Schoellkopf,
SILICIEL, Contributions à l'architecture des circuits intégrés et à la compilation du silicium, thèse,
Université Scientifique et Médicale, Institut National Polytechnique, Grenoble, Avril 1985.

(146) J.P.Schoellkopf,
LUBRICK: A Silicon Assembler and its Application to Data-Path Design for FISC,
VLSI'83,F.Anceau and E.J.Aas Editors,
Elsevier Science Publications B.V. (North Holland)
IFIP, 1983, pp.435-445.

(147) W.S.Scott, J.K.Ousterhout,
PLOWING: Interactive Stretching and Compaction in MAGIC,
21st Design Automation Conference, 1984, pp.166-172.

(148) W.S.Scott, J.K.Ousterhout,
Magic's Circuit Extractor,
22nd Design Automation Conference, 1985, pp.286-292.

(149) H.E.Shrobe,
Abstraction and Layering in Silicon Compilation Tools,
NATO Advanced study Institute on Logic Synthesis and Silicon Compilation for VLSI Design, SSGRR - L'Aquila (Italy), July 7-18, 1986.

(150) H.E.Shrobe,
The Data Path Generator,
Conference on Advanced Research in VLSI, M.I.T., 1982,
pp.175-181.

(151) N.Singh,
MARS:A Multiple Abstraction Rule-Based Simulator, technical report 17,
Fairchild Laboratory for Artificial Intelligence Research, 1983.

(152) J.M.Siskind, J.R.Southard, K.W.Crouch,
Generating custom high performance VLSI designs from succinct algorithmic descriptions,
Conference on Advanced Research in VLSI, M.I.T., 1982, pp.28-40.

(153) P.Six, K.Croes, L.Rijnders, H.De Man,
CAMELEON: a technology independent symbolic layout system,
Rapport IMEC 1985, pp.124-129.

(154) P.Six, I.Vandeweerd, H.De Man,
An Interactive Environment for Creating Module Generators,
ESSCIRC'86, 1986, pp.65-67.

(155) J.A.Solworth,
GENERIC: A Silicon Compiler Support Language,
23rd Design Automation Conference, 1986, pp.524-530.

(156) R.F.Sproull, R.F.Lyon,
The Caltech Intermediate Form for LSI Layout Description,
Caltech, 1980.

(157) Status Report of the Graphic Standards Planning
Committee, Computer Graphics, 13-3, 1979.

(158) Suwardi Iping Supriana,
Mécanismes prédictifs d'évaluation des caractéristiques
géométriques des circuits VLSI, thèse,
Institut National Polytechnique Grenoble, Juin 1985.

(159) Survey of IC Layout CAD Systems,
VLSI Systems Design Staff,
VLSI Systems Design, September 1986, pp.67.

(160) K.S.B.Szabo, M.I.Elmasry,
SYMPLE: a process independant symbolic layout tool for
bipolar VLSI,
IEEE, 1984, pp.474-479.

(161) A.A.Szepieniec,
SAGA: an experimental silicon assembler,
19th Design Automation Conference, 1982, pp.365-370.

(162) Tak-Kwong Ng, S.Lennart Johnsson,
Generation of layouts from MOS circuit schematics: a graph
theoretic approch,
22nd Design Automation Conference, 1985, pp.39-45.

(163) G.S.Taylor, J.K.Ousterhout,
Magic's Incremental Design-Rule Checker,
21st Design Automation Conference, 1984, pp.160-165.

(164) J.H.Tomkinson,
UCAD: Building Design Automation with general purpose
software tools on UNIX,
20th Design Automation Conference, 1983, pp.774-787.

(165) S.Trimberger,
Combining Graphics and a Layout Language in a Single
Interactive System,
18th Design Automation Conference, 1981, pp.234-239.

(166) S.Trimberger,
The Proposed Sticks Standard,
Caltech Computer Science Department, Technical Report 3380,
1980.

(167) S.Trimberger,
RIOT--A Simple Graphical Chip Assembly Tool,
19th Design Automation Conference, 1982, pp.371-376.

(168) S.Trimberger,
VTIcompose - A Powerful Graphical Chip Assembly Tool,
ICCAD'84, 1984, pp.233-235 / 21st Design Automation Conference, 1984, pp.697-698.

(169) S.Trimberger, J.Rowson, C.Lang, J.Gray,
A Structured Methodology and Associated Software Tools,
IEEE trans. on Circuits and Systems, v.CAS-28, n.7, July 1981, pp.618-633.

(170) J.D.Ullman,
Computational Aspects of VLSI,
Computer Science Press, 1984.

(171) UNIX programmer's manual,
UNIX 4.2 BSD, 4th Berkeley Distribution,
University of California, January 18, 1983.

(172) W.M.VanClemput,
Hierarchical Design for VLSI: Problèmes and Advantages,
Proceedings of Caltech Conference on VLSI, January 1979.

(173) A.Vandemeulebroecke, L.Terés, C.Trullemans,
PUDIC: Pascal Used for the Design of Integrated Circuits,
pp.711-714.

(174) N.van der Meijs, R.Nouta,
Procedural Layout Generation and Silicon Assembly in C,
Soumis ESSIRC'85, 1985.

(175) S.Van Vlierberghe, J.Rijmenants, W.Heyns,
Symbolic Hierarchical Artwork Generation System,
22nd Design Automation System, 1985, pp.789-793.

(176) VAX/VMS V3.0,
Command Language,
User's Guide, May 1982.

(177) V.V.Venkataraman, C.D.Wilcox,
GEMS: an automatic layout tool for MIMOLA schematics,
23rd Design Automation Conference, 1986, pp.131-137.

(178) B.Vergnieres,
Macro Generation Algorithms for LSI Custom Chip Design,
IBM Journal of research and development, v.24, n.5, 1980.

(179) C.L.Wardle, C.R.Watson, C.A.Wilson, J.C.Mudge, B.J.Nelson,
A Declarative Design Approach for Combining Macrocells by Directed Placement and Constructive Routing,
21st Design Automation Conference, 1984, pp.594-601.

(180) H.Wertz,
Intelligence Artificielle, Application à l'Analyse de Programmes,
Etudes et Recherches en Informatique, Masson, Paris, 1984.

(181) N.Weste,
Virtual Grid Symbolic Layout,
18th Design Automation Conference, 1981, pp.225-233.

(182) N.Weste and B.Ackland,
A Pragmatic Approch to Topological Symbolic IC Design,
VLSI'81, J.Gray editor, Academic Press, New York, 1981,
pp.117-129.

(183) T.Whitney, C.Mead,
POOH: A Uniform Representation For Circuit Level Designs,
VLSI'83,F.Anceau and E.J.Aas Editors,
Elsevier Science Publications B.V. (North Holland)
IFIP, 1983, pp.401-411.

(184) R.Widdowson,
An Investigation into Stretchable Cells for Scale, internal
report CSR-181-85,
Department of Computer Science, University of Edinburgh,
Scotland, January 1985.

(185) R.Wilensky,
LISPCraft,
W.W.Norton and Company, New York, 1984.

(186) J.D.Williams,
STICKS: A New Approch to LSI Design,
MIT MSEE Thesis, 1977.

(187) J.D.Williams,
STICKS: A Graphical Compiler for High-Level LSI Design,
Proceeding of the National Computer Conference, 1978, pp.289-
295.

(188) J.A.Wilmore,
The Design of the Database for Sphinx IC Artwork Editing and
Verification System,
ICCAD'84, 1984, pp.111-113.

(189) W.Wolf,
An Object-Oriented, Procedural Database for VLSI Chip
Planning,
23rd Design Automation Conference, 1986, pp.744-751.

(190) G.Wood, Hung-Fai Stephen Law,
SKILL-An Interactive Procedural Design Environment,
IEEE 1986 Custom Integrated Circuits Conference, pp.544-547.

(191) J.M.Wrigh, M.S.Fox,
SRL 1.5 User Manual,
CMU Robotics Institute, 1983.

(192) R.V.Zara, D.R.Henke,
Building a layered data base for design automation,
22nd Design Automation Conference, pp.645-651, Juin 1985.

(193) G.Zintl
A Codasyl CAD data base system
18th Design Automation Conference, pp.589-594, Juin 1981.

TABLE DES MATIERES

OUTILS DE CAO ET CONCEPTION STRUCTUREE DE SYSTEMES INTEGRES SUR SILICIUM

INTRODUCTION

PREMIERE PARTIE

METHODOLOGIE DE CONCEPTION, ETAT DE L'ART

DEUXIEME PARTIE

STATION DE CONCEPTION CAO-VLSI

1 - Problèmes liés aux outils de CAO-VLSI

1.1 - Problèmes liés aux actions de conception

1.2 - Intégration des outils de conception

1.2.1 - Propriétés d'un processeur

1.2.2 - Choix d'un mode d'intégration

2 - Approche objet de la programmation

2.1 - Introduction

2.1.1 - Contrainte

2.1.2 - Axes de développement

2.1.3 - Méthodologie

2.1.4 - Outils nécessaires à la conception

2.2 - Les langages orientés objets et les langages acteur

2.2.1 - Structuration des connaissances

2.2.2 - Vocabulaire et définitions

2.2.3 - Modes de représentation des données

2.2.3.1 - Les réseaux sémantiques

2.3 - Synthèse

2.3.1 - Les objets

2.3.2 - Les classes d'objets

2.3.3 - La programmation objet

2.3.3.1 - L'attachement procédural

2.3.3.2 - Le message

2.3.3.3 - Le choix des objets

3 - Spécification du coeur d'un système intégré

3.1 - Aperçu des bases de données CAO-VLSI

3.1.1 - Approche traditionnelle

3.2.8.1 - Choix d'une méthodologie

3.2.8.2 - Outils de gestion de la cohérence

3.2.8.3 - Intérêt pour la conception

TROISIEME PARTIE

OUTILS DE GENERATION DE MASQUES

1 - Vue d'ensemble

1.1 - Noyau des outils de conception

1.2 - Outils de génération des masques

1.2.1 - Structuration des circuits

1.2.2 - Description des blocs

1.2.3 - Cohérence des différentes descriptions

1.2.4 - Approche symbolique

1.2.4.1 - Générateurs et langage procédural

1.3 - Environnement et outils de génération des masques

2 - Corps d'une cellule

2.1 - Génération automatique

2.2 - Génération semi-automatique

2.3 - Génération manuelle

2.4 - Processeurs qui en découlent

3 - Symbolique métrique

3.1 - Spécification

3.1.1 - Principe

3.1.2 - Les articulations

3.1.3 - Les segments

3.1.4 - Les zones

3.1.5 - Facette métrique des cellules

3.2 - Problème technologique

3.2.1 - Base technologique

3.2.2 - Différentes règles

3.2.3 - Indépendance technologique

3.2.3.1 - Symboles et règles technologiques

CONCLUSION

ANNEXE I

REFERENCES

TABLE DES MATIERES

INPG département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 juillet 1984 relatif aux études doctorales

VU les rapports de présentation de Messieurs

. F. ANCEAU Ingénieur d'Etudes
.J.P MOREAU

Monsieur ROUGEAUX François-René

est autorisé à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité "Microélectronique".

Fait à Grenoble, le 19 janvier 1987

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

RESUME

La représentation symbolique des masques de circuits intégrés est connue depuis environ quinze ans, mais n'est développée que depuis quelques années. Presque tous les systèmes de CAO développés récemment font usage d'une représentation symbolique des masques de circuits afin de simplifier la conception.

Dans cette thèse, un système symbolique hiérarchique de dessin des masques est décrit. Un langage de description et un environnement graphique sont présentés. L'environnement graphique est utilisé pour l'édition de cellules et de blocs, hiérarchiquement. Le langage textuel peut être considéré comme le langage cible de tous les outils de synthèse ou de compilation qui sont développés dans le cadre d'un système intégré de CAO VLSI.

Une représentation métrique des circuits est donnée, et un "Assembleur de Silicium", qui permet le dessin hiérarchique de cellules fonctionnelles en accord avec les structures d'interconnexions de base, est décrit. Par un programme Lisp le concepteur définit comment les cellules feuilles doivent être assemblées pour réaliser la fonction désirée. Ainsi des cellules fonctionnelles peuvent être générées, avec une représentation symbolique et une fonctionnalité déterminée.

MOTS CLEF

Systèmes CAO, Bases de données, Systèmes symboliques, Description procédurale, Assembleur de silicium, DRC, Editeur, Simulateur.