N° d'ordre : 511

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le grade de

DOCTEUR EN ELECTRONIQUE

par

Laurent GUILBERT

Ingénieur ISEN

CONTRIBUTION A DES OUTILS DE CONCEPTION DE CIRCUITS INTEGRES

Soutenue le 27 Avril 1990, devant la commission d'examen :

Président	CONSTANT	Université Lille 1
Rapporteurs	DERRYCKE	CUEEP – Lille
	DECARPIGNY	ISEN – Lille
Directeur de Thèse	CONSTANT	Université Lille 1
Examinateurs	RAMBERT	RTC COMPELEC – Caen
	RACZY	USTL – Lille 1
	PETITPREZ	ISEN – Lille

Mener à bien un travail de thèse n'est pas chose facile. Mais la tâche aurait été impossible sans l'aide de toutes les personnes de mon entourage. Je les en remercie vivement, qu'elles soient citées ou non dans cette page.

L'ISEN a mis à ma disposition tous les moyens matériels et humains nécessaires pour mener à bien ce travail. Plus particulièrement, le département informatique et son responsable Mr PETITPREZ m'ont aidé et soutenu tout au long de ces années.

Mr CONSTANT, Président du jury, a eu la patience d'encadrer mes travaux, et m'a servi de lien permanent avec l'Université des Sciences et Techniques de Lille Flandres-Artois.

Mms DECARPIGNY et DERRYCKE par leurs critiques m'ont permis une meilleure formulation de mes idées. Tous deux m'ont fait le plaisir d'être rapporteur officiel.

Mr RACZY a accepté le rôle d'examinateur.

Mr RAMBERT, malgré la distance a accepté le rôle d'examinateur.

Christelle SOYEZ a assuré la mise en page de ce rapport.

SOMMAIRE

LA CONCEPTION DES CIRCUITS INTEGRES

PAGES

L'EDITEUR GRAPHIQUE

LE VERIFICATEUR DE REGLES DE GARDE

L'EXTRACTEUR DE SCHEMAS

REFERENCES

ANNEXES

INTRODUCTION

Ce travail a pour sujet la conception assistée par ordinateur appliquée au domaine de la synthèse des circuits intégrés, numériques ou analogiques.

La réalisation de tels circuits comporte plusieurs phases :

* *L'étape de conception* :

Elle consiste en la définition des spécifications et une étude de faisabilité.

* *L'étape de réalisation* :

Celle-ci se décompose en plusieurs niveaux :

- la réalisation des masques,
- les vérifications,
- l'inversion de schémas.

La réalisation des masques est une étape longue et fastidieuse dont le but est de dessiner l'ensemble des couches constituant le circuit désiré. A partir de ces dessins, le fabricant peut piloter les différents processus industriels conduisant au produit fini.

La vérification des masques est une opération manuellement impossible. En effet, suite à l'étape précédente, on se trouve en possession de schémas constitués de millions d'entités graphiques élémentaires (rectangles, triangles, ...). Ces éléments doivent respecter des contraintes géométriques liées à la technologie de fabrication. Un non respect des règles peut entrainer des disfonctionnements graves du circuit ou des chutes de rendement en production.

L'inversion des schémas est une étape permettant de vérifier que le produit final réalise bien la fonction désirée. En effet, lors du passage de la représentation électrique à la représentation graphique du circuit, des erreurs peuvent être commises provoquant la génération d'un composant non conforme. Une vérification de la cohérence des différentes représentations permet d'éviter un passage inutile en production, opération très couteuse.

De nombreux logiciels d'aide à la conception de circuits intégrés existent sur le marché. Nous pouvons classer ces produits en deux grandes catégories :

* *Les logiciels d'aide à la conception de circuits intégrés numériques.*

Ces produits utilisent au départ une représentation algorithmique de haut niveau décrivant le fonctionnement du circuit. De cette description, par des moyens plus ou moins automatiques, une découpe en blocs fonctionnels est réalisée. Ces blocs sont ensuite placés et connectés. Cette approche ressemble assez fortement à la méthode de conception d'un circuit imprimé. Pour de tels logiciels, les étapes de vérification et d'inversion de schémas ne sont pas nécessaires, ces contraintes étant respectées à la fabrication.

* *Les logiciels d'aide à la conception de circuits intégrés analogiques.*

On ne trouve ici aucun produit complet, mais simplement divers logiciels réalisant des fonctions bien précises et ayant des interfaces plus ou moins normalisées entre eux. Dans ce domaine, les efforts ont essentiellement portés sur les phases de dessins des masques et de vérification des masques.

Différents problèmes existent pour chacune de ces catégories de logiciels. Pour la première, le problème majeur est celui de la surface occupée par le circuit réalisé. Ce point est génant dans la mesure où le prix de revient est proportionnel à la surface. De plus, ces produits ne permettent pas la réalisation de circuits analogiques. Pour la seconde catégorie, l'inconvénient majeur vient du manque d'unité de l'offre logicielle. De plus, ces programmes ne permettent généralement pas la conception de circuits numériques nécessitant une structuration des données plus rigoureuse et formalisée.

Notre but était dans cette étude, de trouver un compromis entre ces deux approches et d'offrir un ensemble cohérent de logiciels permettant la réalisation de circuits intégrés numériques et analogiques. Ceci c'est fait par la réalisation de logiciels suffisamment performants pour traiter le numérique, ayant suffisamment de souplesse pour autoriser les réalisations analogiques et parfaitement interfacés entre eux. De plus, nous tenions à faire fonctionner ces logiciels sur du matériel moyenne gamme afin de pouvoir généraliser ces techniques.

Le travail présenté dans ce rapport vise à atteindre au mieux ces objectifs.

Ce document est composé de quatre parties.

- *La conception de circuits intégrés.*

  Une analyse des différentes méthodes et logiciels disponibles sur le marché dans le domaine considéré permet de justifier nos axes de développement.

- *L'éditeur graphique.*

  Le logiciel présenté permet de faire de la schématique pour circuits imprimés, circuits intégrés analogiques et circuits intégrés numériques.

- *Le vérificateur de règles de garde.*

- *L'extracteur de schémas.*

  Une simplification des modèles permet d'obtenir des résultats satisfaisants sans nécessité la puissance de calcul d'un super-ordinateur.

  Un couplage serré avec l'éditeur de masques, permettant un gain de temps important, est justifié au travers des fonctionnalités du produit.

LA CONCEPTION DES CIRCUITS INTEGRES

Dans ce chapitre, les différentes possibilités s'offrant aux concepteurs pour la réalisation des circuits intégrés sont présentées par un rapprochement de méthodes de conceptions et des outils disponibles actuellement sur le marché. Les points faibles sont énoncés.

Le plan du chapitre est le suivant :

- Les étapes de réalisation d'un circuit intégré, qui fait état des différentes méthodologies de conception existantes.

- Les logiciels nécessaires, présentant les outils nécessaires et leurs fonctionnalités.

- Récapitulatif, effectuant la relation entre les besoins et les outils et indiquant les manques.

- La position des travaux réalisés, présentant les différents points traités dans l'étude et justifiant nos choix.

I- LES ETAPES DE REALISATION D'UN CIRCUIT INTEGRE

La complexité et la taille des circuits intégrés ne cessent d'augmenter. Ce développement a nécessité la mise en oeuvre de méthodes de travail et d'outils informatiques qui facilitent le travail de conception. Parmi ces méthodes, on peut citer en particulier la conception hiérarchique des circuits [1] [2].

En effet, pour la phase initiale, le concepteur fait une analyse descendante (TOP-DOWN). Il considère d'abord le système complet, puis le découpe en sous-ensembles de plus en plus simples, jusqu'à arriver aux éléments de base. Par contre, pour l'implantation, la démarche du concepteur sera ascendante (BOTTOM-UP). Il créera d'abord des cellules de base qu'il assemblera pour former des blocs plus complexes, et ceci jusqu'à définir le circuit complet.

Quant aux outils, ce sont essentiellement des programmes de conception assistée par ordinateur, utilisant fortement le graphique [3]. Ce besoin se fait sentir à tous les niveaux : architecture, conception logique, conception électrique, "layout", vérification, test, mais aussi dans les liaisons entre ces différentes étapes de la conception d'un circuit.

1) LA CONCEPTION

Les étapes initiales de la réalisation d'un circuit sont la définition du problème, l'analyse des besoins et la décomposition du système en sous-systèmes. Les fonctionnalités du circuit et les contraintes à respecter (vitesse, coût...) sont établies pendant la phase de définition des besoins.

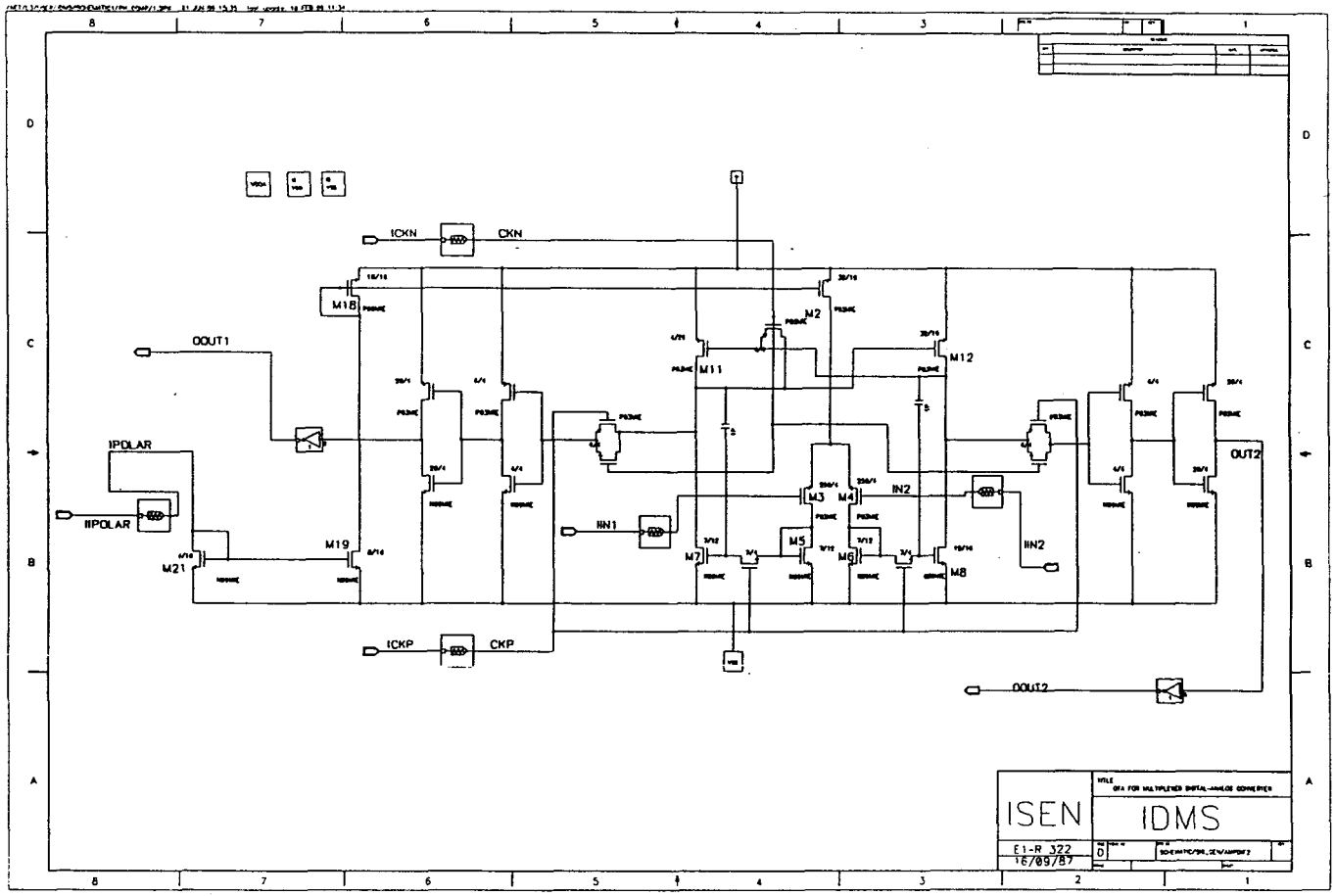
La définition finale du circuit nécessite de faire des compromis entre les contraintes et les besoins. De cette phase découle la structure du système qui est ensuite découpée en sous-systèmes interconnectés avec des interfaces clairement définies. Pour les grands circuits, les sous-systèmes peuvent être considérés comme des systèmes à part entière et faire l'objet du même travail de définition.

Ce n'est que très récemment (début des années 80) que l'automatisation de cette phase a donné naissance à des outils opérationnels. Ces outils sont essentiellement des langages de description tels que VHDL, en cours de normalisation.

Cette première phase de conception d'un circuit intégré aboutit à l'élaboration d'un ou deux schémas du circuit :

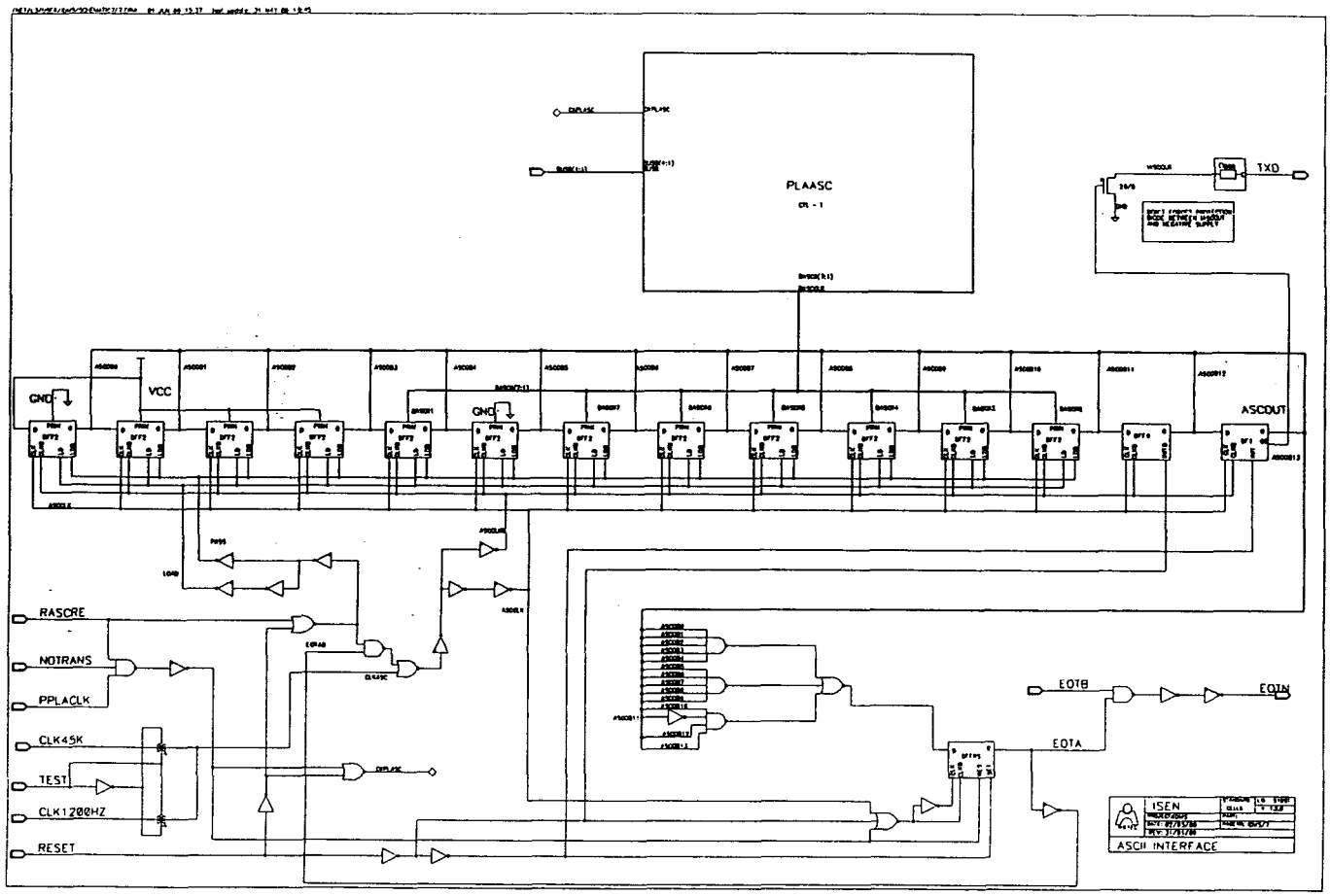
- un schéma électrique où les entités de base sont des transistors et des composants passifs (résistances, capacités...) comme dans l'exemple donné en figure 1. Ce schéma sera implanté sur silicium.

Figure 1 :



- un schéma logique où les entités de base sont des éléments logiques (portes, registres, multiplexeurs ...) comme dans l'exemple donné en figure 2. Ce schéma doit par la suite être transformé en schéma électrique, de manière manuelle ou automatique. Cette étape n'a pas lieu d'être dans la réalisation d'un circuit analogique.

Figure 2 :



2) LES SIMULATIONS

Sur ces schémas (ou descriptions) sont effectuées des simulations qui permettent au concepteur de vérifier la validité de son étude théorique.

La simulation fonctionnelle permet de vérifier que le système réalise correctement les fonctions définies avant de commencer une conception plus fine. Le résultat de cette simulation est aussi utilisé pour la génération des tests.

A ce niveau, le concepteur doit fournir l'algorithme décrivant le fonctionnement de son circuit. A cause du coût de fabrication, la détermination de la testabilité du circuit doit se faire avant de commencer la conception détaillée. Il faut être sûr de pouvoir tester le circuit après la fabrication.

Le but est de produire des circuits où toute erreur potentielle peut être isolée en appliquant des entrées connues au circuit et en observant les sorties. Cela suppose que toute faute soit contrôlable et observable. Il faut donc que pour chaque faute, il existe une combinaison des entrées faisant apparaître une sortie inexacte.

La simulation logique repose sur une technique d'exécution dirigée par événements. Un tel simulateur peut considérer le temps de 2 façons :

- une succession d'intervalles élémentaires . Ce genre de simulateur est particulièrement inefficace car le temps de simulation dépend de la durée, et non du nombre d'événements. Ceci est particulièrement sensible dans le cas où des circuits rapides (nécessitant un intervalle de temps court) sont pilotés par une horloge lente.

- des intervalles de temps variables déterminés par les portes et les entrées. Dans ce cas, le temps de simulation ne dépend que du nombre de changements d'états à l'intérieur du circuit.

Très souvent, les simulateurs logiques et les simulateurs fonctionnels sont présents dans le même logiciel. Ceci permet de faire de la simulation mixte et d'éviter ainsi une simulation complète du circuit en terme de portes de base (ET, OU, NON, bascules, registres...). C'est la seule façon de simuler entièrement un circuit complexe en conservant des temps de réponse acceptables.

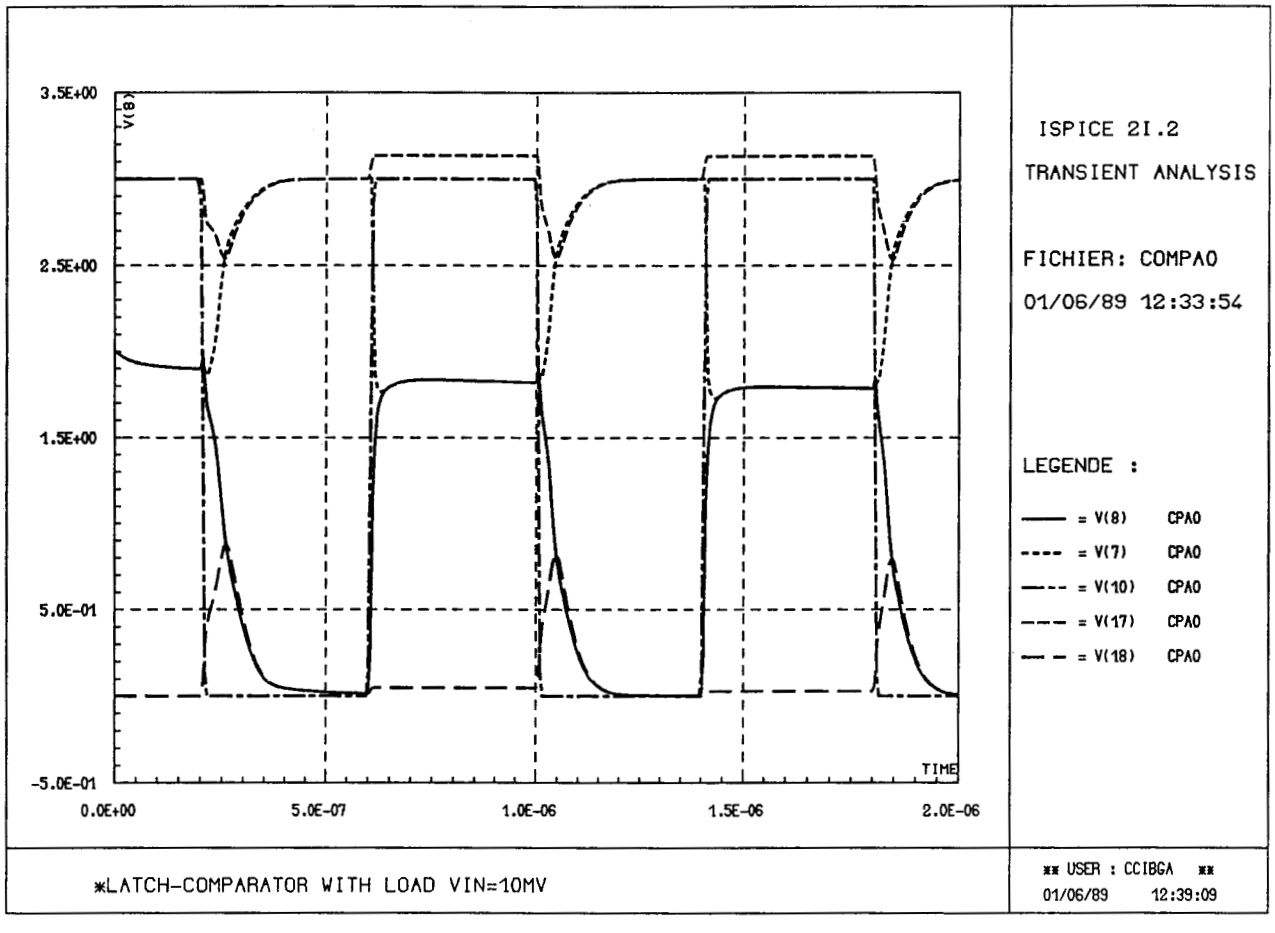
Pour la simulation au niveau "switch", chaque transistor est modélisé comme un interrupteur contrôlé par une tension. Trois états différents sont utilisés :

OFF (circuit ouvert),
ON (circuit fermé),
R (résistif).

Cette simulation n'est utile que si l'on s'intéresse au comportement logique ou temporel du circuit. Les résultats sont généralement proches de ceux produits par une simulation électrique détaillée.

La simulation électrique travaille au niveau des transistors et des composants actifs ou passifs. Elle est rarement effectuée sur l'ensemble du circuit, mais plus généralement sur les chemins critiques. Le but de cette simulation est de fournir des courbes de fonctionnement du circuit telles que celles-ci :

Figure 3 :



Elle doit aussi intégrer les paramètres de la technologie utilisée. Cette contrainte est primordiale pour la validité des résultats de simulation et pose beaucoup de problèmes au concepteur. En effet, la plupart des simulateurs offrent peu de possibilités de modification des caractéristiques des composants. Pour connaître ces paramètres, deux solutions sont envisageables :

- une simulation fine du process technologique permet maintenant de déterminer les caractéristiques des composants générés. Malheureusement, peu de fabricants fournissent le détail de leur technologie pour des raisons évidentes de protection.

- La deuxième solution consiste à réaliser un circuit test, sur lequel on effectue des mesures précises des caractéristiques. On adapte ensuite les paramètres de description des composants dans le simulateur, afin d'obtenir la meilleure correspondance possible entre le réel et le théorique. Généralement, ces itérations sont réalisées par un programme qui prend le simulateur en boucle. Il faut noter que cette méthode est gourmande en ressource machine.

3) L'IMPLANTATION

Une fois achevée la conception détaillée du circuit, l'étape suivante consiste à réaliser le dessin des masques (ou "layout") qui seront utilisés pour la fabrication du circuit.

Cette phase consiste à placer des milliers (millions) de rectangles ou polygones sur une large surface. Le placement de ces rectangles définit les éléments fonctionnels du circuit tels que les transistors et les interconnexions. De plus, la largeur et l'espacement de ces rectangles doivent respecter un certain nombre de contraintes fonction de la technologie afin d'éviter des erreurs de fabrication.

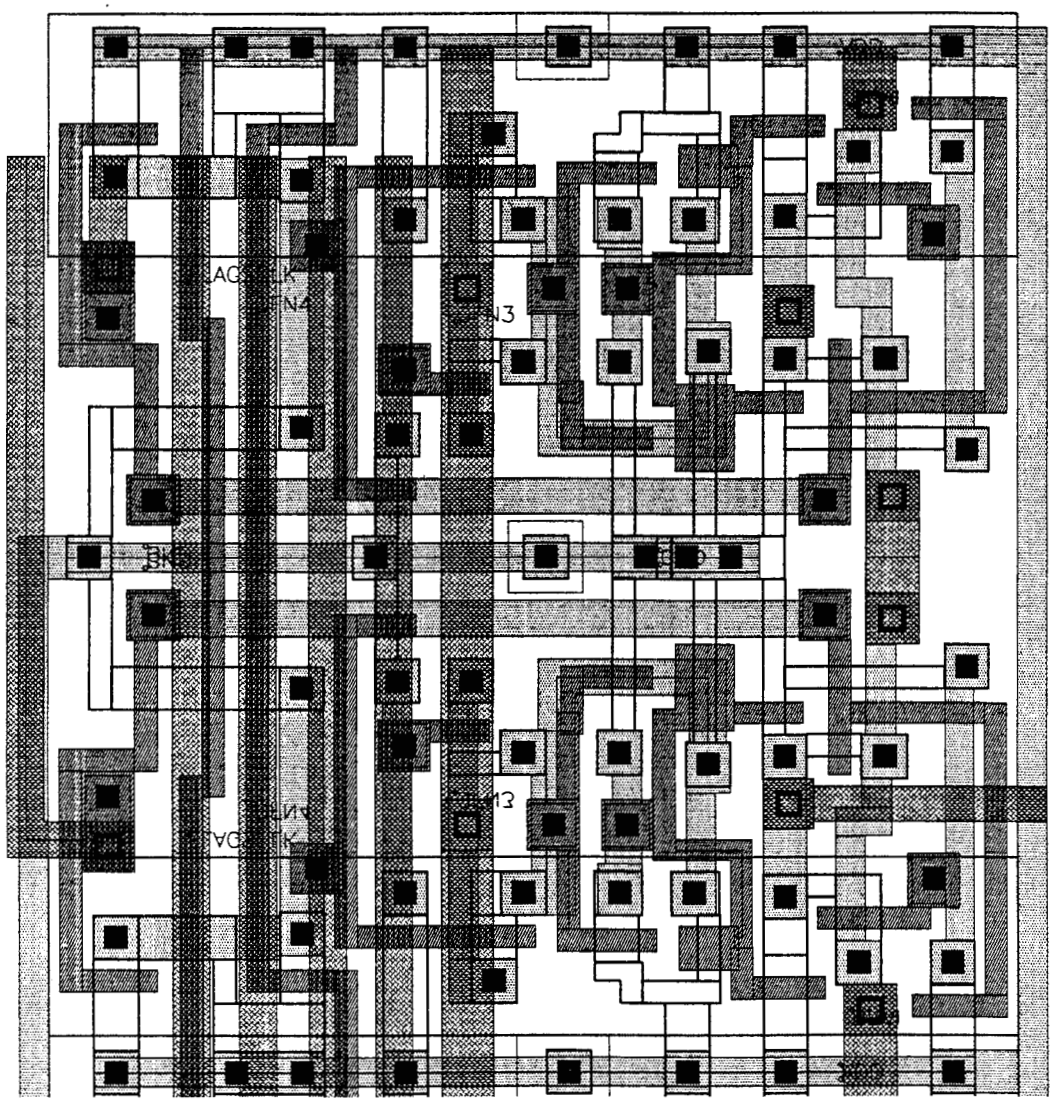
Cette étape peut être longue et fastidieuse ou très rapide suivant la méthode employée. Par contre, la surface du circuit résultant, donc le coût, est généralement inversement proportionnel au temps passé. Les techniques existantes visent toutes un seul objectif : minimiser le temps de réalisation du layout pour une surface de circuit.

Les méthodes les plus courantes sont :

- Le dessin manuel :

Dans ce cas, il existe peu ou pas de structures prédéfinies. Le concepteur commence par la réalisation de ses cellules de base, puis il les interconnecte pour réaliser des cellules plus grandes qu'il assemble pour former des blocs encore plus grands, et ainsi de suite jusqu'à la réalisation du circuit complet. Cette méthode donne le meilleur résultat sur le plan de la place occupée, et donc du coût de fabrication, mais est la plus coûteuse en ressources humaines : voici un exemple de layout manuel.

Figure 4 :



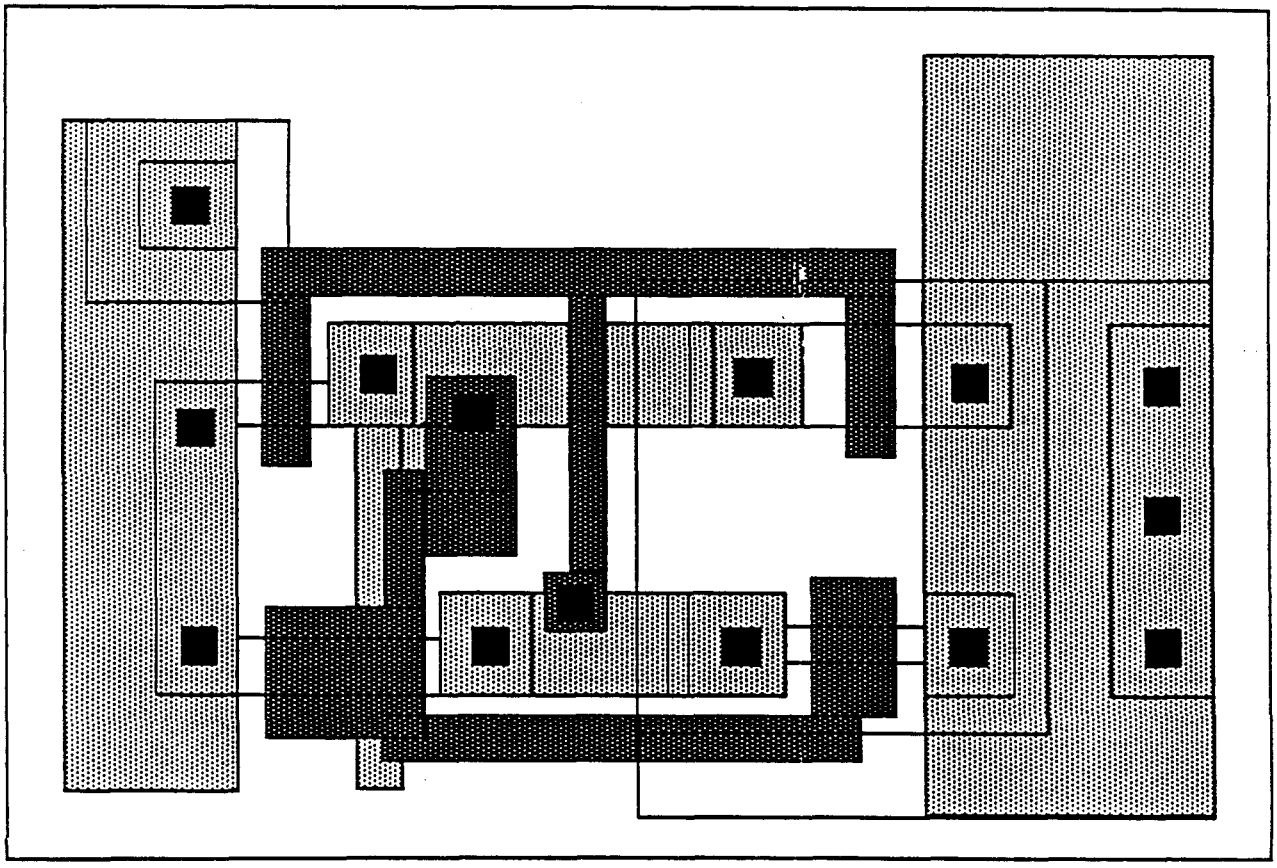
- Le dessin symbolique :

dans ce cas, le concepteur introduit également des données de façon manuelle, mais il ne génère que la topologie du dessin : on dessine librement des lignes de largeur nulle ; ces lignes remplacent les pistes ; les transistors et les contacts, représentés par des symboles, sont placés aux intersections de ces lignes. Une fois la topologie déterminée, un programme transforme le dessin en fonction des règles technologiques et on obtient un "layout" assez dense. Plusieurs systèmes de ce genre existent : STICK [4] et CABBAGE [5] .

- Les cellules standards :

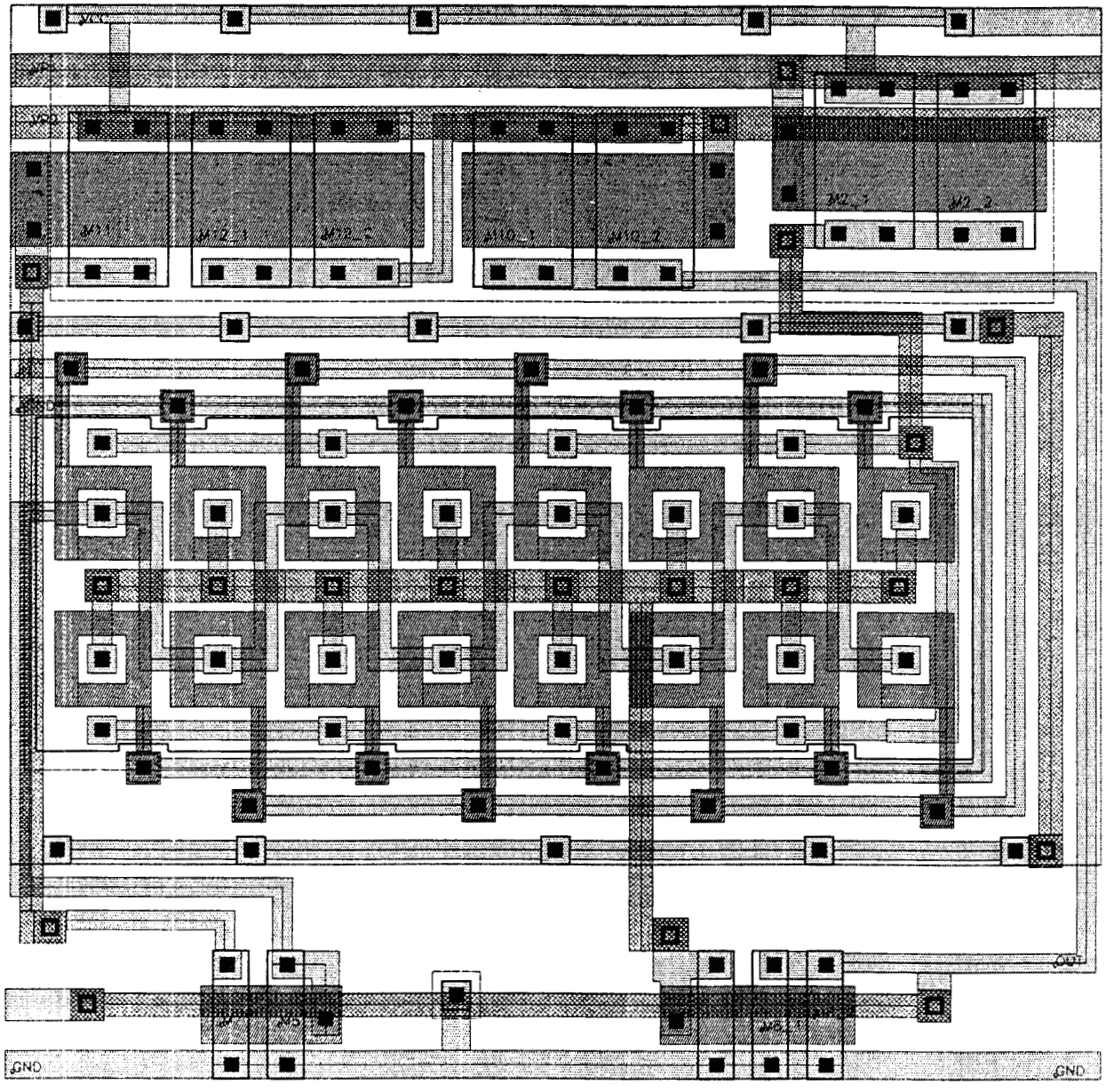
le concepteur dispose d'une bibliothèque de cellules de base dont le layout est optimisé. Toutes les cellules sont contenues dans un rectangle dont l'une des dimensions est une constante. Chaque cellule réalise une fonction et elle est traversée par au moins deux bus, un pour la Masse et un pour l'Alimentation : voici un exemple de cellule standard.

Figure 5 :



Le concepteur assemble ces cellules par rangées ou colonnes et effectue ensuite les interconnexions. Il lui est possible d'utiliser des logiciels de placement et de routage pour la réalisation de son circuit. Le gain de temps est important, mais la perte de place est non négligeable, principalement à cause du routage canal utilisé dans cette méthode. On peut citer deux systèmes basés sur cette approche : CALMOS [6] , LTX [7] : voici un exemple de circuit obtenu par cette méthode.

Figure 6 :



- Les compilateurs de silicium :

la méthode développée par Johansen s'applique à tous les niveaux de la conception, depuis l'architecture jusqu'au "layout" ([8] , [9]). On n'évoquera ici que ce dernier. Des cellules sont conçues pour être accolées deux à deux. Elles sont paramétrables : selon les paramètres spécifiés, on obtient pour une même cellule, différentes vitesses de fonctionnement, différentes puissances consommées.

La cellule peut être agrandie suivant une dimension : il est ainsi possible de relier directement les sorties de cette cellule avec les entrées de celle qui lui est accolée. Johansen a appliqué cette méthode pour réaliser la partie opérative, souvent appelée "data path", d'un microprocesseur. Cette technique semble donc très intéressante et pourra sûrement remplacer dans l'avenir le dessin manuel avec de bons résultats.

- Les "gate arrays" :

ici, on utilise des circuits préfabriqués contenant des transistors ou des cellules réalisant des fonctions très simples, et des lignes d'alimentation sans aucune connexion. Le fabricant peut donc produire des quantités importantes de circuits non routés, et faire la connexion à la demande du client. Comme la fabrication des couches de connexions est fiable et bon marché, les prix sont tirés vers le bas. De plus, la mise en place d'un seul niveau de connexion est peu coûteuse en ressources machine. Cette méthode est donc très rapide. Par contre, la place occupée est très moyennement optimisée.

4) LES VERIFICATIONS

La vérification constitue la dernière étape de conception du layout. Celui-ci comporte plusieurs milliers (millions) d'éléments graphiques de base (rectangles, lignes, polygones, trapèzes...). Les processus de fabrication imposent des contraintes géométriques sur ces différents éléments, tels que :

- largeur minimale des lignes
- espacements entre deux éléments d'un même niveau
- recouvrement minimal entre éléments de niveaux différents ...

Un exemple de toutes les règles à vérifier sur une technologie NMOS est donné en annexe 1.

La quantité d'informations à traiter est telle qu'une vérification manuelle est impossible. L'existence de vérificateurs de garde est donc indispensable.

Une deuxième vérification importante est la cohérence entre le circuit qui a été simulé et le circuit obtenu effectivement (oubli d'un contact, mauvais dimensionnement d'un transistor...). Ici aussi, une vérification manuelle est impossible. Il est donc nécessaire d'extraire le schéma réellement implanté au niveau du circuit à partir du dessin des masques.

Après extraction, il devient possible, par l'intermédiaire d'un outil de comparaison, de valider le schéma réel du circuit. Un autre avantage de cette étape est de déterminer, en fonction de l'implantation, les éléments parasites tels que les résistances, les capacités, les diodes qui seront implicitement ajoutées à la fabrication. De cette extraction fine, il est possible de simuler les chemins critiques et par conséquent de valider de façon exacte le comportement du circuit.

II- LES LOGICIELS NECESSAIRES

Nous reprenons ci-après les différentes étapes de réalisation d'un circuit et pour chacune d'elles, les logiciels nécessaires et les fonctions qu'ils doivent réaliser.

1) LA CONCEPTION

Cette phase est essentiellement du domaine du concepteur. A ce niveau, la seule aide informatique envisageable est une aide au dessin. Les logiciels utilisés sont donc des logiciels de "schématique" dont la seule fonction est une aide à la réalisation et à la maintenance des schémas. Il existe maintenant quelques logiciels d'aide à la conception. Venues du monde du logiciel, des méthodes telles que SADT (Structured Analysis and Design Technique) permettent de décrire le flot des données et des signaux de contrôle à l'intérieur d'un système.

Ces outils sont tout à fait adaptés à la conception de circuits intégrés car ils ont été créés pour décrire le fonctionnement de systèmes complexes. De plus, ils sont indépendants de la technologie. Quelques approches théoriques telles que les réseaux de PETRI ont été développées pour la description de systèmes asynchrones, mais à l'heure actuelle, peu de choses sont disponibles.

Le département américain de la défense a mis au point un langage de description standard VHDL (VHSIC Hardware Description Language) qui permet de décrire un système et ses contraintes en termes de structure et de fonctionnalités. La description d'un additionneur 1 bit en VHDL est la suivante :

```
architecture GATE_VIEW of FULL_ADDER is view :
        block
        - Local component declarations
        component AND_GATE port(A,B:in BIT;C:out BIT) ;
        component XOR_GATE port(A,B:in BIT;C:out BIT) ;
        component OR_GATE  port(A,B:in BIT;C:out BIT) ;
        - Local signal declarations
        signal S1,S2,S3:BIT ;
        begin
        - Component instantiations
        X1:XOR_GATE port(X,Y,S1) ;
        X2:XOR_GATE port(S1,Cin,SUM) ;
        A1:AND_GATE port(Cin,S1,S2) ;
        A2:AND_GATE port(X,Y,S3) ;
        O1:OR_GATE port(S2,S3,Cout) ;
        end block ;
end GATE_VIEW ;
```

On peut remarquer que cette description est très proche de la programmation en langage ADA.

Une autre approche pour aider le concepteur est de lui fournir des outils de simulation et de synthèse travaillant au niveau système.

Par exemple, lors d'une première analyse, le concepteur se pose des questions telles que : combien de mémoires faut-il, à quelle vitesse doit fonctionner la machine ?

Pour évaluer ces besoins, deux méthodes existent.

- La simulation du système :

  - Ecrire un simulateur dédié
  - Utiliser un simulateur existant tel que N.2 [10] qui permet de faire des simulations à ce niveau de conception.

- La compilation de silicium :

  - Approche structurelle
  - Approche fonctionnelle

La plupart des développements de circuit occasionne l'écriture d'un simulateur dédié. Cette méthode présente l'inconvénient de ne pas pouvoir explorer rapidement plusieurs architectures. L'utilisation de simulateurs tels que N.2 supprime ce problème.

La compilation de silicium existe sous 2 formes. Dans l'approche structurelle, le concepteur décrit chaque bloc du système en fournissant des informations telles que la taille des mots, les contraintes temporelles et les interconnexions.

A partir de ces informations, le compilateur génère le layout qui peut être simulé et fabriqué. Cette approche impose une description d'assez bas niveau. Dans l'autre approche, la compilation de silicium commence par une description fonctionnelle du circuit et produit automatiquement le layout. La simulation est faite au niveau fonctionnel. Malheureusement, les produits de cette nature sont encore en laboratoire, sauf un : MetaSyn de Metalogic.

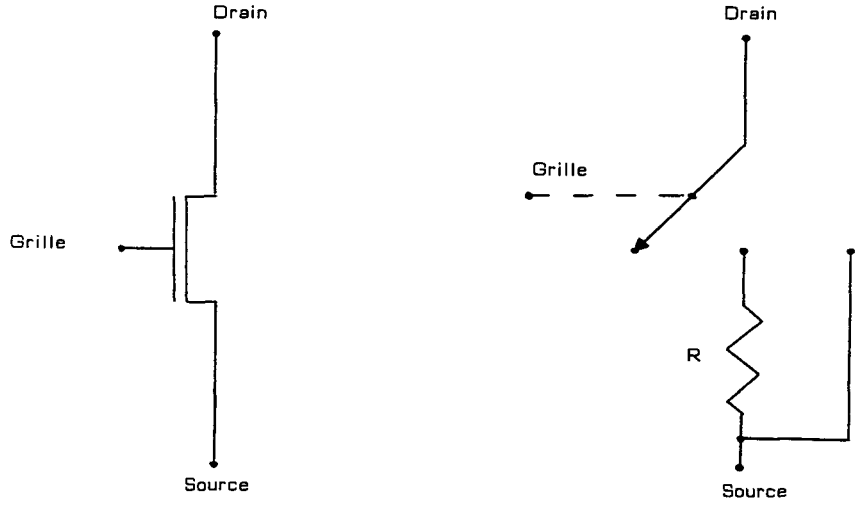
2) LES SIMULATIONS

La simulation est de plus en plus liée à la conception. En effet, il paraît logique d'offrir au concepteur la possibilité de valider son travail au fur et à mesure de son avancement. C'est pourquoi la plupart des logiciels de schématique offre des interfaces plus ou moins évoluées avec les simulateurs. Ceci permet d'éviter une description supplémentaire du circuit dans le langage propre au simulateur.

On effectue en général une simulation électrique détaillée des chemins critiques ou des cellules afin d'obtenir des informations précises sur les temps de propagation et le fonctionnement de ces parties. Ces simulations sont rarement effectuées sur des circuits de plus de 100 transistors pour des problèmes de temps de calcul. Le plus connu des simulateurs électriques est SPICE [12], qui permet la modélisation de composants en tenant compte des paramètres technologiques tels que la capacité de grille ou la mobilité des électrons.

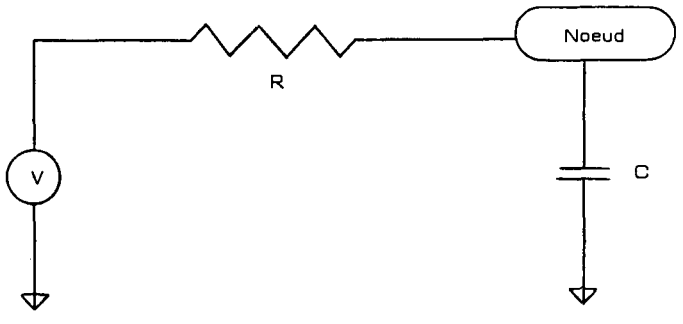
La simulation au niveau "switch" permet de valider le fonctionnement global du circuit. Dans des produits comme RSIM [14] développé à Carnegie-Mellon, les transistors sont modélisés de la façon suivante :

Figure 7 :



Dans ce modèle, l'impédance drain-source est 0, 1 ou R en fonction de la position du switch. Chaque noeud du réseau a le circuit équivalent suivant :

Figure 8 :



V, R et C sont calculés à partir des paramètres du transistor et des tensions appliquées au noeud. RSIM a modélisé des circuits comportant jusqu'à 400.000 transistors.

Les simulateurs logiques tels que HILO de GENRAD acceptent en entrée différentes représentations du circuit. On peut mixer des descriptions fonctionnelles, au niveau portes de bases (ET, OU, NON, registres...) et au niveau circuits prédéfinis (circuits de base tels que ceux des familles 74LSXX, AMDXXX ou autres). Il est aussi possible, pour des raisons de performance, de lui adjoindre un simulateur "hardware", où on utilise le composant réel, et non une description de son fonctionnement. Dans ces logiciels, des modules de calcul de la testabilité d'un circuit et de la détermination des données de tests sont en général disponibles.

L'utilisation de ces produits de simulation entraîne un besoin important en ressource machine. En effet, l'espace disque et la mémoire centrale nécessaires sont importants. Pour HILO par exemple, un espace disque de 15 méga-octets est le minimum requis pour pouvoir stocker le produit et la bibliothèque de base. Pour certains produits de CAO comme CBDS d'IBM, la bibliothèque occupe un espace disque de plus de 60 méga-octets. De façon générale, la mémoire centrale utilisée par ces produits est de l'ordre de 6 méga-octets. De plus, ces logiciels réalisent énormément d'entrées-sorties, ce qui impose l'utilisation de disques et de bus très rapides.

En conséquence, une station de travail haut de gamme (SUN 4, VAXSTATION 3000 ...) est la machine minimale pour obtenir un temps de traitement acceptable par le concepteur. Il faut cependant noter que ces simulateurs fonctionnent sur presque toutes les machines de grande diffusion, ce qui permet à l'utilisateur de ne pas être lié à un constructeur.

3) L'IMPLANTATION

Cette étape a pour but de fournir les masques du circuit au fabricant. Un layout peut être très complexe, et les concepteurs doivent utiliser ordinateurs et graphisme pour avoir de l'aide. La plupart des concepteurs utilise des systèmes graphiques interactifs pour réaliser les masques. Le layout se construit de façon purement hiérarchique, en construisant des cellules de plus en plus complexes [15]. Partout où cela est possible, des structures régulières telles que les mémoires ou les PLA sont utilisées de préférence à la logique "sauvage", ceci afin de réduire la complexité de la conception. Des éditeurs graphiques tels que MAGIC [13] de l'université de Californie, permettent de créer des composants ou des cellules sur un écran graphique couleur.

Quand les composants sont créés et interconnectés, ils sont stockés dans une base de données, puis simulés et fabriqués. Ce logiciel offre des fonctionnalités supplémentaires. A chaque fois qu'un nouvel objet est ajouté à l'écran, une vérification des règles de garde est effectuée. Si une erreur de dessin existe, l'utilisateur est immédiatement prévenu à l'écran. Une autre fonction intéressante de ce produit, est le rétablissement automatique des connexions quand le concepteur bouge un composant ou une cellule. De plus, MAGIC est capable de faire du placement et du routage automatique.

Les besoins varient beaucoup en fonction de la méthode utilisée. Ils peuvent aller de la nécessité d'un "simple" éditeur graphique (dessin manuel), à l'emploi d'un ensemble complet de conception (compilateur de silicium). Il faut cependant noter que la réalisation d'un circuit analogique impose quasiment de faire un dessin manuel, les efforts d'automatisation de la conception de circuits intégrés étant très faibles dans ce domaine. Seuls les circuits logiques sont actuellement du ressort des compilateurs de silicium.

4) LES VERIFICATIONS

Pour cette phase, deux fonctionnalités différentes sont nécessaires :

La vérification des règles de garde nécessite peu ou pas de connaissance des différents composants du circuit. Dans cette étape, les masques sont manipulés comme un ensemble d'éléments graphiques qui doivent respecter un certain nombre de contraintes géométriques (voir annexe 1 pour une technologie NMOS). Afin d'être indépendant de la technologie employée, le logiciel sera constitué d'un ensemble d'opérateurs de masques. Le concepteur devra créer un fichier contenant la suite des opérations à effectuer, ainsi que les opérandes à manipuler. Les opérateurs de ce logiciel sont des opérateurs de vérification de garde intérieure ou extérieure mono ou multi masques, et un ensemble de fonctions logiques telles que le ET, le OU, le NON... .

Les fonctions logiques permettent d'extraire des sous-ensembles du layout pour lesquels existent des contraintes spécifiques (débordement de 4 μm du polysilicium au delà du transistor dans la technologie NMOS 6 μm). Les algorithmes du DRC (Design Rule Checking) nécessitent un temps d'exécution proportionnel à la complexité dans le meilleur des cas, et à la complexité au carré dans le plus mauvais. La complexité d'un layout peut s'exprimer de différentes manières, soit comme étant le nombre de rectangles (ou figures élémentaires) du dessin, soit comme étant le nombre de transistors du circuit. Nous retiendrons la première définition qui est indépendante de la technologie employée.

Après l'étape de dessin des masques, il est important de réaliser une simulation du circuit complet pour vérifier son bon fonctionnement. Les éléments tels que les transistors et les interconnexions doivent donc être extraits afin de permettre une simulation au niveau "switch". L'extraction de schémas impose de retrouver les composants constituant le circuit et leurs interconnexions. Les opérateurs de ce logiciel sont les opérateurs logiques du produit précédent pour la reconnaissance des éléments (par exemple, pour une technologie NMOS, un ET entre le polysilicium et la diffusion donnera les transistors), un ensemble de fonctions permettant de déterminer les différentes équipotentielles du circuit (ce qui nous donnera les interconnexions), ainsi que les composants qui s'y rattachent. A partir de ces renseignements, on peut générer la liste des noeuds (Netlist).

De nombreux programmes existent. Un exemple typique est ACE [11], développé à l'université de Carnegie-Mellon. Pour ce produit, on peut estimer un temps de traitement de 220 minutes sur un VAX 11/780 pour le traitement d'un circuit de 100.000 transistors. Ce résultat est obtenu par interpolation linéaire de résultats de traitements moins importants. Il est à noter que 40% du temps est employé pour le tri et la préparation des données. Ce facteur temps est aussi valable pour le DRC. Peu de logiciels réalisent la comparaison de deux netlists. Il est plus courant de faire une simulation du circuit extrait et de vérifier son bon fonctionnement, plutôt que de vérifier la cohérence entre le schéma désiré et le schéma implanté.

III- RECAPITULATIF

Quels sont les outils informatiques disponibles pour la conception des circuits intégrés ?. Le tableau suivant présente les principales tâches qui peuvent être simplifiées par les outils de CAO dans le cadre de la réalisation d'un circuit intégré logique.

TACHE	OBJECTIFS DE LA TACHE	ACTEUR
conception générale	description du circuit au niveau bloc	concepteur
simulation fonctionnelle	simulation du circuit décrit comme un ensemble de registres et de fonctions logiques	logiciel
conception détaillée	description de chaque sous système au niveau portes logiques ou transistors	concepteur
simulation logique	simulation du circuit décrit comme un ensemble de portes et de bascules	logiciel
évaluation de la testabilité	évaluation du taux de testabilité du circuit	logiciel
génération des tests	génération de listes d'entrées de tests et des sorties attendues pour le test du circuit après fabrication	logiciel
dessin des masques	transformation du schéma logique détaillé en masques utilisés pour la fabrication	concepteur logiciel
vérification des règles de garde	vérification que toutes les contraintes géométriques de largeur, espacements et recouvrements sont corrects pour tous les masques	logiciel
extraction de schéma	extraction d'un schéma au niveau transistors à partir des masques	logiciel
simulation au niveau "switch"	simulation du circuit extrait en considérant tous les transistors comme des transistors de passage	logiciel
simulation du circuit	simulation de tout ou partie du circuit au niveau composant	logiciel

Beaucoup d'outils de CAO ont pour but d'accélérer la conception et la validation. Cependant, personne (à l'exception d'IBM et son système LDS), ne prend en compte la standardisation du format des données et des langages de description. Il en est de même pour les problèmes d'archivage de l'énorme quantité de données générées pendant la création d'un circuit. En effet, chacun de ces logiciels possède en général son propre mode de stockage des informations. Ceci conduit à une multiplicité des bases de données, ce qui entraîne des risques d'incohérence entre ces différents modes de représentation des informations.

IV- LA POSITION DES TRAVAUX REALISES

L'étude décrite dans le document ci-après porte sur les étapes d'implantation et de vérification. Son but a été d'analyser les algorithmes envisageables pouvant fonctionner sur du matériel peu coûteux. Nous avons restreint le champ de l'étude au cas de circuits intégrés analogiques, car dans le domaine de la logique, de nombreux produits existent, comme nous l'avons vu précédemment.

Le premier centre d'intérêt a porté sur la réalisation d'un éditeur graphique pour les étapes de saisie de schémas et de dessin des masques. Notre objectif était de montrer qu'un seul logiciel pouvait remplir correctement les deux fonctions. Nous ne nous sommes pas intéressés à la simulation. Les produits existent, et peu de choses innovantes restent à faire de ce côté.

Notre deuxième centre d'intérêt a porté sur les phases de vérification, que ce soit la vérification des règles de garde ou la validation du circuit implanté. Pour le DRC, nous sommes partis du constat que la majeure partie d'un layout est constitué de rectangles. Aussi, il nous a semblé intéressant d'utiliser le rectangle comme entité de base. L'étude a prouvé que même si ce choix n'est pas parfait, le résultat était intéressant et que la piste méritait d'être creusée.

Pour l'extraction de schémas, nous voulions savoir si une comparaison de netlists n'était pas plus intéressante ou plus performante qu'une simulation du circuit obtenu au niveau "switch". De plus, les produits existants n'extraient que les transistors et les interconnexions. Nous avons prouvé qu'il est possible de retrouver les éléments parasites induits par la fabrication.

Nous avons choisi ces produits car ils sont réputés consommer beaucoup de place mémoire et d'espace disque, et nécessiter une puissance de calcul importante. Notre étude a montré que ces logiciels, moyennant des algorithmes adaptés, peuvent fonctionner sur du matériel de type PC, avec des temps de réponse tout à fait acceptables. De plus, dans le cas des circuits intégrés , ces étapes prennent énormément de temps. Par conséquent, il est intéressant de ne mobiliser que du matériel peu coûteux, offrant malgré tout un confort d'utilisation acceptable par le concepteur.

L'EDITEUR GRAPHIQUE

Les concepts mis en oeuvre pour la réalisation d'un éditeur graphique pour circuits intégrés sont présentés.

Le "gigantisme" des données manipulées ainsi que la limitation des outils existants contraignent à mettre en oeuvre des structures et bases de données complexes. Néanmoins, cette complexité interne n'est pas visible pour l'utilisateur, grâce à une ergonomie soignée.

Le plan du chapitre est le suivant :

- Les objectifs visés, rappelant l'objet du travail.

- La norme graphique GKS, (Graphic Kernel System) présentant les avantages et inconvénients des outils existants à ce niveau.

- Les données manipulées, leur "gigantisme" et leur complexité.

- La structure d'informations, permettant de résoudre les problèmes lié à l'utilisation de GKS.

- La base de données, stockant dans un espace minimal les informations nécessaires à la description d'un circuit.

- Quelques fonctions disponibles, montrant qu'une ergonomie soignée peut cacher la complexité interne d'un logiciel.

I- INTRODUCTION

La plupart des logiciels de C.A.O proposés actuellement sur le marché présente l'inconvénient de ne pas être portables, cette caractéristique entraîne inévitablement les inconvénients suivants :

- Une dépendance vis-à-vis du constructeur du matériel informatique.

- Une difficulté d'évolution en puissance de calcul, les produits n'étant pas toujours disponibles sur l'ensemble de la gamme d'un constructeur. Par exemple IBM, qui commercialise des outils CAO sous VM, alors que ce système ne sait pas exploiter les capacités des plus grands ordinateurs du constructeur. Ce problème est en cours de résolution.

- le concepteur doit dans certains cas connaître plusieurs produits différents bien que ceux ci présentent les mêmes fonctionnalités. En effet, peu de fournisseurs proposent actuellement la schématique logique, électrique et le dessin des masques en un seul produit.

La plupart des produits de C.A.O ont été développés initialement sur de gros systèmes, par la force des choses puisque seules ces machines étaient suffisamment puissantes à l'époque. L'apparition des stations de travail, au cours de ces dernières années, a entraîné une évolution très nette des mentalités et des techniques. L'existence de micro-ordinateurs de faible coût doit maintenant conduire à proposer des produits utilisables sur l'ensemble de la gamme, même si l'utilisation de systèmes moins performants entraîne une certaine perte de confort. Cet inconvénient ne devrait plus en être un très longtemps étant donnée la montée en puissance des PC ou des PS qui ont maintenant des performances qui sont tout à fait comparables à celles des stations de travail.

II- LES OBJECTIFS VISES

1) LA DESCRIPTION HIERARCHIQUE

Un des buts recherchés est d'offrir à l'utilisateur un produit s'adaptant à sa méthode de conception. Celle-ci étant hiérarchique, notre produit le sera aussi. De plus, le concepteur peut réaliser son étude de façon descendante ou ascendante en fonction de son avancement. Cela nous a conduit à créer un logiciel où la hiérarchie peut être manipulée dans les deux sens, et ceci de manière conviviale.

2) LA PORTABILITE

La portabilité d'un logiciel graphique sur des systèmes provenant de plusieurs constructeurs peut se résoudre à condition de prendre en compte les points suivants :

- choix du langage de programmation
- choix des normes graphiques

Le langage de programmation retenu est le FORTRAN 77 en raison de sa disponibilité sur l'ensemble des systèmes informatiques ayant une vocation de calcul scientifique. Ce choix peut apparaître surprenant en raison de l'existence de langage tel que PASCAL ou C. Cependant, il faut noter que PASCAL n'est pas encore normalisé, et que C ou ADA ne sont pas implantés sur tous les systèmes (notamment sur les machines les plus puissantes).

L'indépendance vis-à-vis des équipements graphiques utilisés est garanti en s'appuyant sur des normes graphiques internationales (GKS, PHIGS). L'utilisation de ces bibliothèques normalisées impose quasiment l'utilisation de FORTRAN, qui est le seul interface fourni en standard sur tous les environnements. Des interfaces avec le langage C commencent à apparaître sur tous les systèmes, mais n'étaient pas disponibles au début de notre étude.

3) LA BASE DE DONNEES

Le logiciel s'appuie sur une base de données qui permettra le stockage de tous les schémas. Afin de garantir une indépendance vis-à-vis d'autres produits nécessaires dans les différentes phases de conception (simulation, vérification, extraction...), l'éditeur graphique possède son propre format. Ceci peut paraître contraignant, mais c'est la seule solution pour garantir la portabilité. Cette base est constituée d'un ensemble de fichiers standards du FORTRAN, ne posant aucun problème de migration. De plus, le fait d'avoir une base de données propre à l'éditeur était indispensable pour pouvoir stocker à la fois les schémas logiques, les schémas électriques et les dessins de masques. Nous avons en effet généralisé notre éditeur afin d'offrir un interface graphique unique au concepteur, ce qui ne peut que lui faciliter la tâche. Le rôle du concepteur n'est pas d'apprendre à se servir d'outils informatiques, mais de réaliser un circuit. Dans ce cas, l'informatique doit être un outil et non un but.

4) LA COMMUNICATION AVEC L'EXTERIEUR

Pour garantir la communication avec l'extérieur, le produit offre des interfaces dans les formats GDS2 (Graphic Data Stream) et CIF (Caltech Intermediate Format) qui sont les deux représentations les plus utilisées pour l'échange de données en CAO électronique. Ces interfaces fonctionnent dans les deux sens et permettent donc de lire et d'écrire du GDS2 et du CIF.

III- LA NORME GRAPHIQUE GKS

1) LES STATIONS DE TRAVAIL

Au sens GKS (Graphic Kernel System), on appelle station de travail (workstation) tout périphérique sur lequel on peut faire des tracés et/ou des entrées sorties.

Ces stations peuvent être de différents types :

- écrans graphiques
- traceurs
- fichiers (métafichiers)

Plusieurs stations peuvent être ouvertes simultanément, mais une seule est active à un instant donné.

2) LES TRANSFORMATIONS

Trois espaces (surfaces planes abstraites caractérisées par un système de coordonnées) sont définis :

- WC-SPACE : World coordinate space

Système de coordonnées utilisateur avec lequel le programme d'application composera son dessin.

- NDC-SPACE : Normalized device coordinate space

Espace de coordonnées normées qui peut être considéré comme une surface de visualisation abstraite indépendante des postes de travail. Les coordonnées sont comprises entre 0 et 1 dans les deux dimensions.

- DC-SPACE : Device coordinate space

Espace des coordonnées de l'écran. On définit ici la taille de l'écran (ou autre station de travail) utilisé exprimée en mètres.

Il existe aussi 4 régions rectangulaires distinctes :

- NT-WINDOW : Région définie dans le WC-SPACE. C'est dans cette zone que le graphique est tracé. Plusieurs NT-WINDOWS peuvent être définies dans une même application.

- NT-VIEWPORT : Région définie dans le NDC-SPACE. C'est la projection de la NT-WINDOW dans l'espace de coordonnées normalisées. On a une NT-VIEWPORT par NT-WINDOW.

- WORKSTATION-WINDOW : Région définie dans le NDC-SPACE. Tout graphique contenu dans l'intersection de cette fenêtre et de la NT-VIEWPORT sera visualisé à l'écran.

- WORKSTATION-VIEWPORT : Région définie dans le DC-SPACE. Zone de l'écran dans laquelle sera effectivement visualisé le graphique.

Une fois ces 3 espaces et ces 4 fenêtres définies, il ne manque plus que la définition de 2 transformations pour pouvoir travailler :

- Transformation de normalisation

  WC-SPACE ---> NDC-SPACE

  NT-WINDOW ---> NT-VIEWPORT

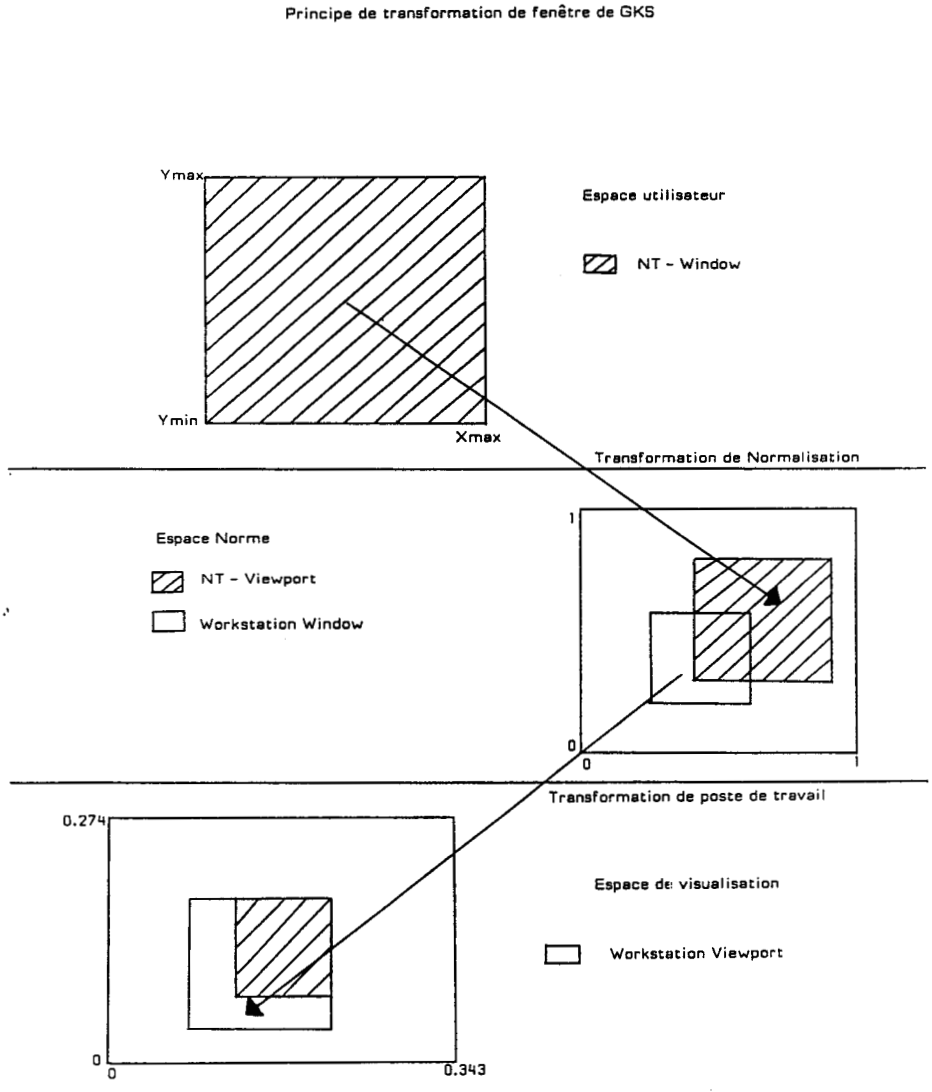
- Transformation de poste de travail

  NDC-SPACE ---> DC-SPACE

  WORKSATION-WINDOW ---> WORKSTATION-VIEWPORT

La figure 9 illustre le principe des transformations des fenêtres de GKS.

Figure 9 :



3) LES PRIMITIVES GRAPHIQUES

On distingue 4 grandes primitives de tracé :

- GPL : Tracé d'un polygone fermé ou non. Les attributs modifiables sont la couleur et le type de tracé (plein, pointillés, tirets...).

- GFA : Tracé d'un polygone fermé avec remplissage. Les attributs modifiables sont le type de remplissage (plein,creux,hachuré...) et la couleur.

- GCA : Tracé d'une matrice de carrés pleins. Ceci permet d'afficher, par exemple, des palettes de couleurs ou des images. Le seul attribut modifiable est la couleur de chaque élément. Tous les carrés constituant la matrice ont la même taille.

- GTX : Tracé d'un texte. Les attributs modifiables sont la couleur, la police de caractères, l'angle d'écriture et le sens d'écriture (haut-bas, droite-gauche...).

4) LA SEGMENTATION

On appelle segment graphique, un ensemble d'entités de base (GPL, GFA, GCA, GTX) regroupées dans un objet que l'on désire manipuler dans son ensemble. Ces segments sont repérés par des numéros. Il est possible de distinguer des sous-parties à l'intérieur d'un segment. Les opérations disponibles sont la translation, la rotation et le changement d'échelle. Toute combinaison de ces opérations est possible. L'utilisateur peut aussi renommer, copier ou insérer un segment.

L'utilisation des segments est indispensable pour la réalisation d'un éditeur graphique, car ce sont les seules entités pour lesquelles on peut définir des attributs de visibilité et de détectabilité, et auxquelles on peut attribuer une priorité d'affichage et de sélection.

5) LES ROUTINES D'INTERACTION

Afin de rendre le logiciel interactif, le programmeur dispose d'un ensemble de fonctions permettant d'interagir avec l'utilisateur. Il est possible de :

- saisir les coordonnées d'un point à l'écran (request locator).

- sélectionner un segment et l'un de ses sous ensembles (request pick).

- demander à l'utilisateur de fournir une chaîne de caractères (request string).

- sélectionner une réponse parmi un ensemble (request choice).

- saisir une suite de points de manière continue (request stroke).

IV- LES DONNEES MANIPULEES

Un circuit intégré est composé de couches superposées dépendantes de la technologie utilisée. Chaque niveau (ou layer) est composé d'un ensemble de contours fermés (rectangles, polygones...). Dans le cas d'une technologie NMOS, on compte 9 niveaux :

- Diffusion : Ce niveau sert à réaliser les source et drain des transistors, mais aussi des interconnexions.

- Transistors naturels : Ce niveau permet d'indiquer les transistors qui ne sont ni enrichis ni déplétés. Ces transistors servent essentiellement à la protection des entrées contre l'électricité statique.

- Implantation ionique : Un transistor qui subit une implantation ionique devient un transistor déplété. Les autres sont enrichis.

- Contact direct : Ce masque permet d'établir des contacts entre le niveau de polysilicium et le niveau de diffusion.

- Polysilicium : Ce niveau sert à réaliser les grilles des transistors, mais aussi des interconnexions.

- Trou de contact : Ce masque permet de réaliser des liaisons entre différentes couches technologiques.

- Surtrou : Ce niveau représente le débordement de métal autour du trou.

- Aluminium : Ce niveau ne sert qu'à réaliser des connexions.

- Passivation : Ce masque représente les zones du circuit qui ne doivent pas être passivées (couvertes de verre). Seuls les plots de contact ne sont pas passivés.

Chacun de ces niveaux peut être composé de plusieurs milliers (ou millions) de figures élémentaires. Le traitement d'une telle quantité de données est impensable sans la mise en oeuvre d'une description hiérarchique du circuit. Ceci n'est pas limitatif dans la mesure où le raisonnement du concepteur, comme nous l'avons vu précédemment, est lui aussi hiérarchique.

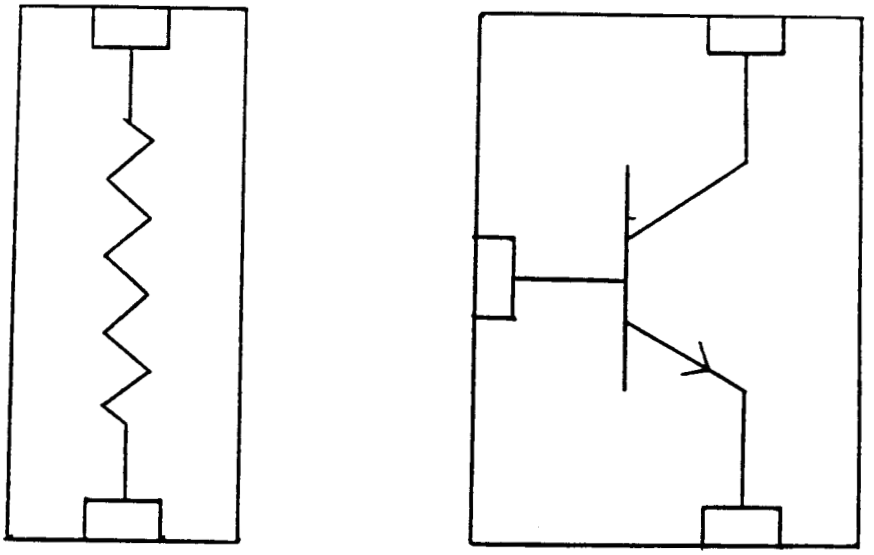
V- LA STRUCTURE D'INFORMATION

1) EXEMPLE DE HIERARCHIE

L'exemple traité ci-après pose correctement la plupart des problèmes de la représentation d'un schéma au sens général.

Soit à représenter la construction d'un étage différentiel. Dans un premier temps, le concepteur doit se définir deux éléments de base : la résistance et le transistor.

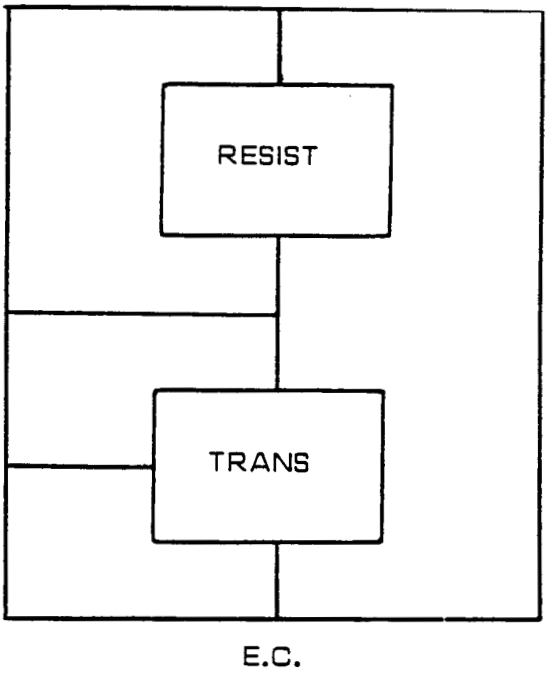
Figure 10 :



Chacun de ces éléments de base occupe une certaine surface décrite par un contour fermé que l'on appelera ENCOMBREMENT. Les différents éléments pourront être interconnectés les uns aux autres par des CONNEXIONS qui réunissent des points d'entrée et de sortie. Ces points seront appelés CONNECTIQUES.

Après création des éléments de base, le concepteur peut réaliser un montage plus complexe : émetteur commun, réalisé par l'assemblage de deux cellules de base.

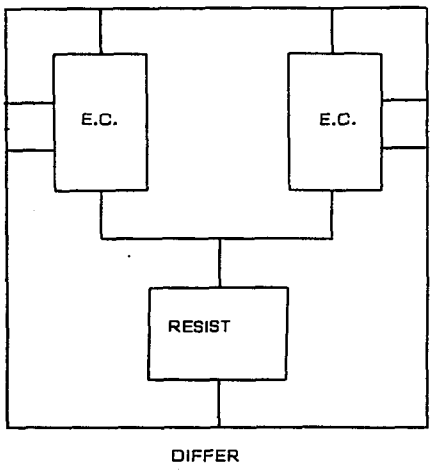
Figure 11 :



Ce circuit est défini par un certain encombrement, ses connectiques, les cellules qui la composent : RESIST et TRANS et les connexions qui réunissent les points d'entrée et de sortie des cellules.

La construction de l'étage différentiel se fait en réalisant une symétrie suivant l'axe des Y pour la cellule émetteur commun et en insérant la cellule "resist". Cette nouvelle cellule comporte son propre encombrement, ses propres connectiques. Elle est constituée de trois cellules "filles" qui sont reliées par des connexions.

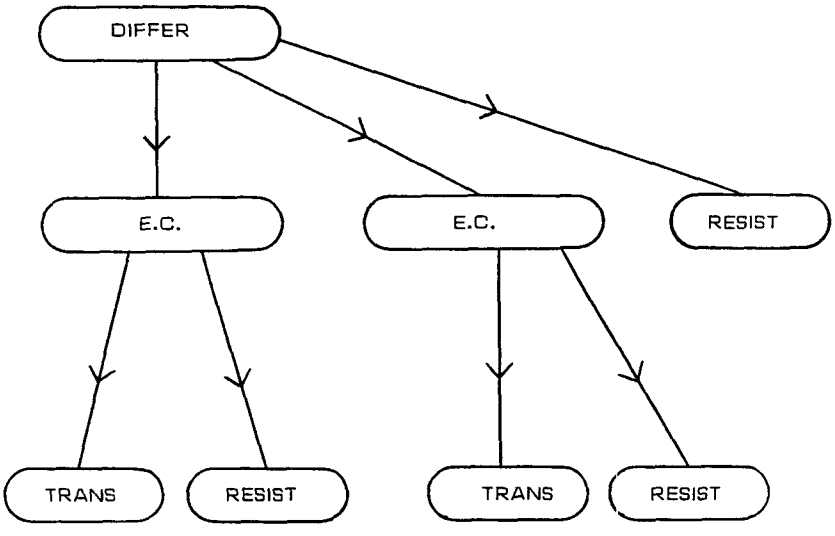
Figure 12 :



Si on ne prend pas certaines précautions sur le plan de la représentation des informations en mémoire centrale, les volumes de données seront vite incompatibles avec des équipements de type micro-ordinateurs. La cellule qui vient d'être définie, comporte certaines cellules "filles" identiques sur le plan du tracé, seule la position origine de ces cellules diffère.

L'exemple traité précédemment introduit une notion de dépendance PERE FILS dans la construction de cellules complexes. Ceci conduit immédiatement à une représentation de type arborescente.

Figure 13 :



2) CONSTRUCTION DE LA STRUCTURE

La description arborescente d'un circuit doit viser plusieurs objectifs :

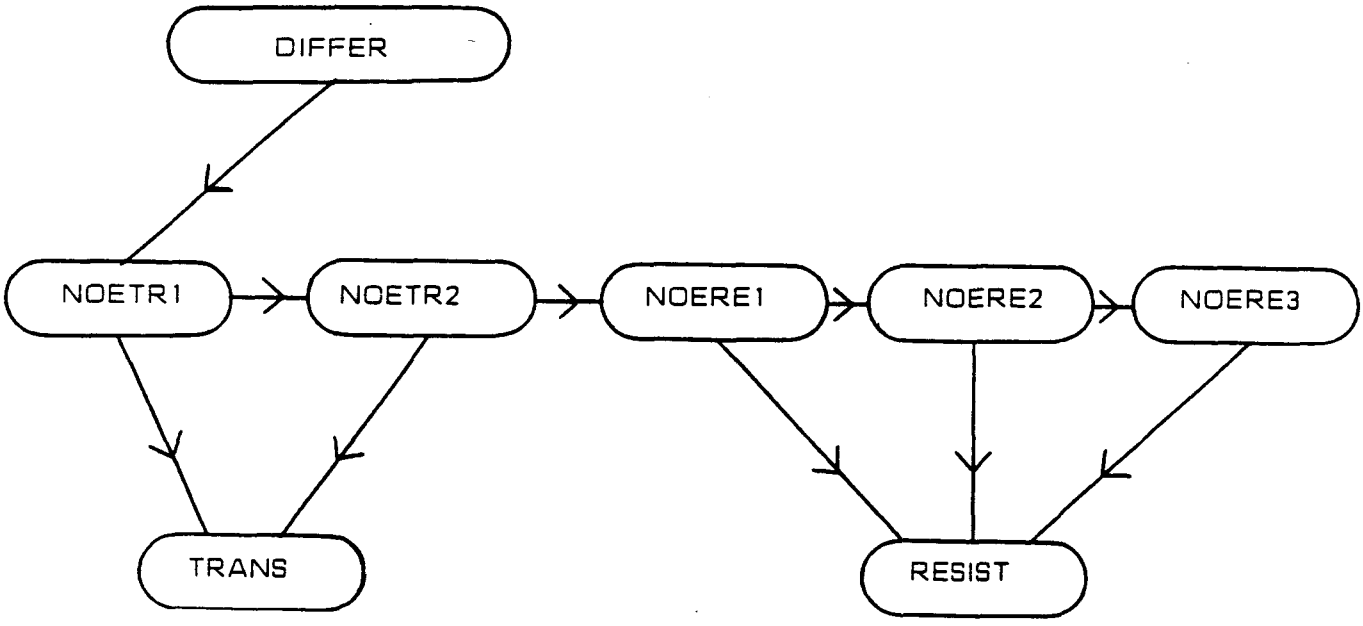
- réduire au maximum la place requise par la structure de donnée. La description géométrique de tout objet identique ne devra être présente qu'une fois et une seule en mémoire centrale.

- Permettre la sélection de tout élément graphique visible à l'écran et pouvoir déterminer le plus rapidement possible l'entité à laquelle il appartient.

Reprenons l'exemple de l'étage différentiel vu au paragraphe E.1. On définit une entité BLOC permettant de contenir une seule fois les figures nécessaires à l'affichage d'un élément (transistor, résistance...). Chaque transistor est défini dans un autre élément de la structure appelé NOEUD qui pointe vers le BLOC correspondant.

Si l'utilisateur construit l'étage différentiel par assemblage direct de 2 transistors et 3 résistances, la structure sera la suivante :

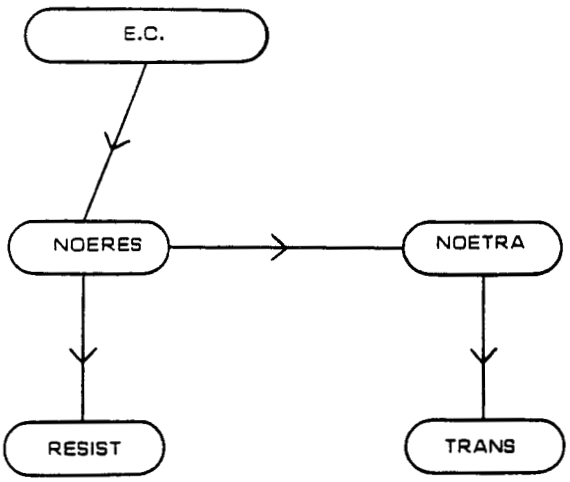
Figure 14 :



Les entités NOExxx contiennent les positions et les opérations (rotations, symétries, translations, répétitions) subies par les dessins qu'elles pointent. Les autres entités contiennent l'ensemble des informations permettant de réaliser le dessin (dimensions, couleur, type de figure...). Ainsi, DIFFER ne contiendra que les lignes servant à lier les résistances et les transistors.

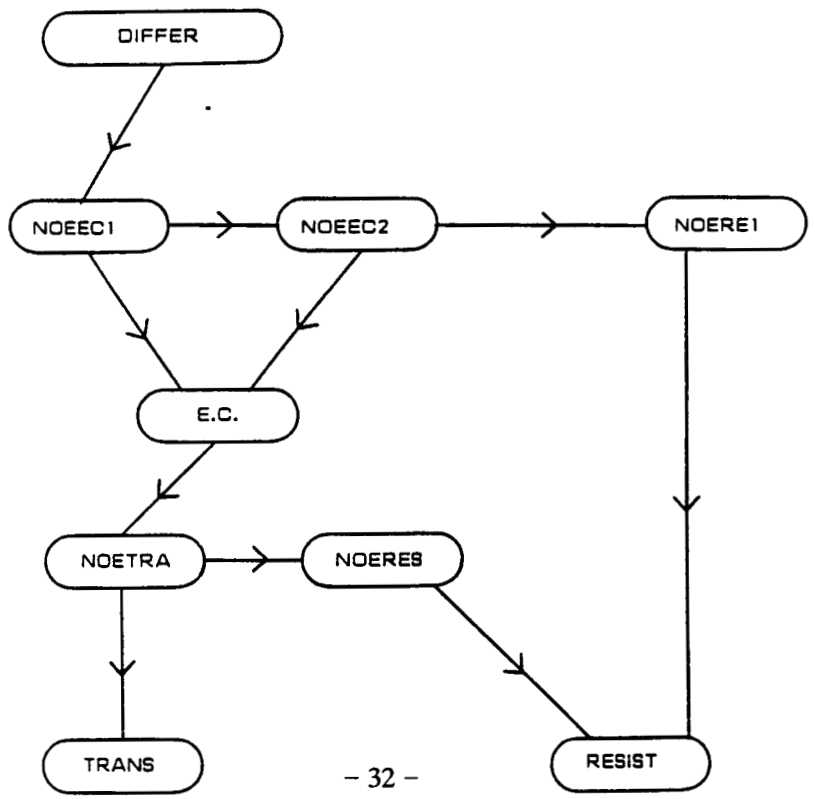
La construction d'un étage émetteur commun donnerait la structure suivante :

Figure 15 :



La construction de l'étage différentiel à partir de l'étage émetteur commun nous donnerait, en suivant cette méthode, la structure suivante :

Figure 16 :



Cette structure ne permet pas de représenter entièrement DIFFER. En effet, on constate qu'il n'y a que deux noeuds pointant sur la résistance et un seul sur le transistor, alors qu'il nous en faudrait un pour chacune des résistances et des transistors du circuit.

On ne pourrait conserver cette structure que pour une arborescence à 2 niveaux.

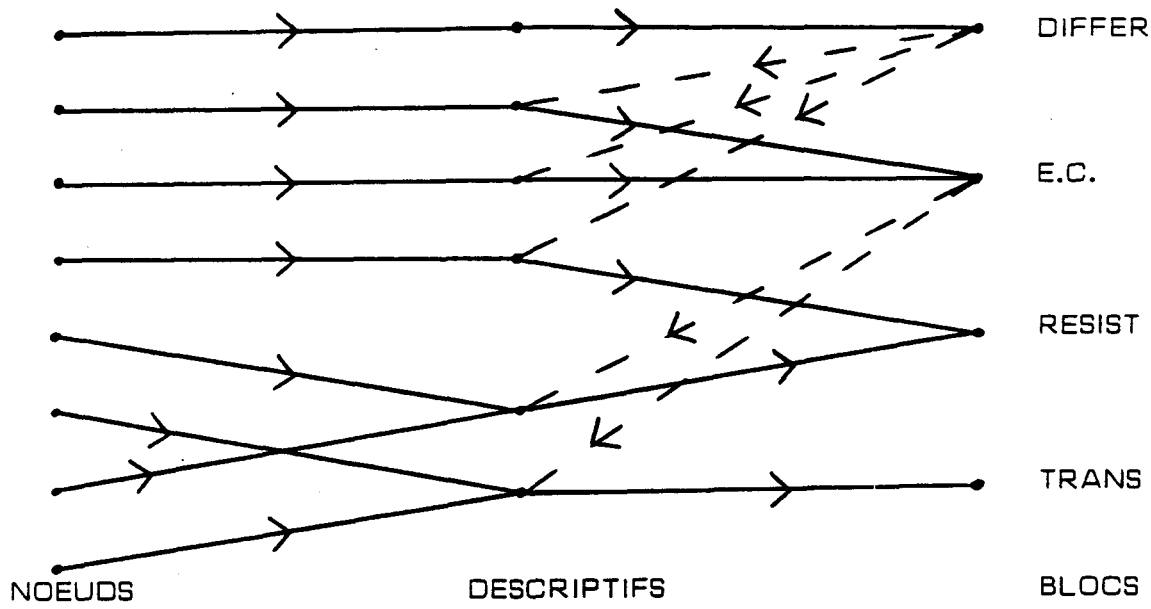
Cette structure étant insuffisante, il convient d'ajouter une nouvelle entité : le DESCRIPTIF, qui contient les informations caractérisant l'utilisation qui est faite d'un bloc (position par rapport au père, symétrie...) dans le cas où ce bloc est utilisé dans la descendance d'un autre bloc. Les noeuds ne sont plus que des entités graphiques représentant les divers éléments du circuit.

Cette nouvelle entité pointe naturellement vers le bloc qu'elle décrit, mais elle est à son tour pointée par le bloc père afin qu'un bloc soit défini par ses figures graphiques et sa descendance.

Les noeuds pointeront eux aussi les blocs qu'ils représentent, mais il faudra auparavant qu'ils aient pris en compte les opérations et positions du bloc. Le pointeur sera donc en direction d'un descriptif, qui pointera le bloc.

L'étage différentiel précédent sera donc obtenu par l'utilisation de 8 noeuds, 6 descriptifs et 4 blocs.

Figure 17 :



— — — : Liaison d'un bloc vers ses descriptifs fils

3) PRESENTATION DE LA STRUCTURE

La structure de données décrite précédemment peut paraître complexe. Une description détaillée est donnée en annexe 2. Cette complexité pourrait être réduite par l'utilisation d'une bibliothèque graphique gérant la hiérarchie, ce qui n'est pas le cas de GKS. PHIGS répond à ce besoin, mais n'est actuellement disponible que dans l'environnement IBM. Reprenons point par point les différents éléments de cette structure, et regardons plus en détail.

- Le catalogue courant : à un instant donné, le concepteur ne dispose que d'une base de données en lecture-écriture. En effet, nous pensons qu'il faut une base par circuit.

- Le catalogue temporaire : le point précédent ne doit pas empêcher l'utilisateur de réutiliser des cellules existantes. Il doit pour cela, copier ces éléments dans sa base. Ceci est logique, puisque la technologie peut changer d'un circuit à l'autre.

- Les formes : ces éléments contiennent les coordonnées des graphiques, sous forme de paires de points.

- Le bloc : cet élément décrit le contenu d'une cellule. Il contient un pointeur vers les formes représentant l'encombrement (ce qui permet de créer des encombrements de forme quelconque, ce qui autorise la schématique logique et électrique) vers les connectiques et connexions du bloc et vers ses descriptifs fils pour la description de la hiérarchie du circuit.

- Le descriptif : il pointe vers le bloc qu'il décrit. Il contient la position de son bloc par rapport au point origine de son père, ce qui autorise des opérations sur des cellules composées. Ainsi les opérations subies par un bloc sont la composition des opérations subies par ses pères et ses propres transformations. Ses fils héritent de ses opérations.

- Le noeud : il décrit l'arborescence visible. A chaque occurrence d'un bloc dans le circuit est associé un noeud, alors qu'une seule entité BLOC existe. C'est le noeud qui permet de faire la relation entre les segments GKS et la structure du circuit (descriptifs et blocs). L'utilisation d'une bibliothèque graphique hiérarchique telle que PHIGS autorise la suppression de cette entité.

- La connectique : cette entité est utilisée lors d'une visualisation de niveau 1 d'un bloc. Dans ce cas, on affiche seulement l'encombrement et la connectique d'un bloc, ce qui permet d'accélérer le chargement du circuit, et allège le schéma. Chaque connectique est nommée (VDD, VSS, IN, OUT...)

- Les connexions : on appelle connexion tout ensemble de formes appartenant à un niveau donné. Ces entités pointent donc vers les formes constituant les cellules de base, mais aussi vers les formes connectant des blocs fils à l'intérieur d'un bloc père.

- Les opérations : ici, on décrit les opérations (rotation, symétrie, translation...) subies par une occurrence d'un bloc.

- Les "segments" : dans ces entités, on effectue la correspondance entre les segments GKS, et les noeuds. Les "segments" (au sens de notre éditeur et non au sens GKS) sont indispensables pour la gestion de l'interactivité graphique. Ces "segments" sont aussi chaînés par niveaux afin de pouvoir rendre un niveau visible ou non.

- Les niveaux : pour chaque niveau de la technologie employée, il faut définir la couleur, le type de tracé et la visibilité.

Cette structure de données est indépendante de la base de données. Ceci est réalisé par l'emploi de fonctions de haut niveau, indépendantes de la structure physique de la base. Ainsi, seuls quelques sous-programmes simples seront à modifier en cas de changement de la base (ou en cas de changement de système, ce qui entraîne toujours quelques problèmes dans la gestion des entrées-sorties).

VI- DESCRIPTION DE LA BASE DE DONNEES

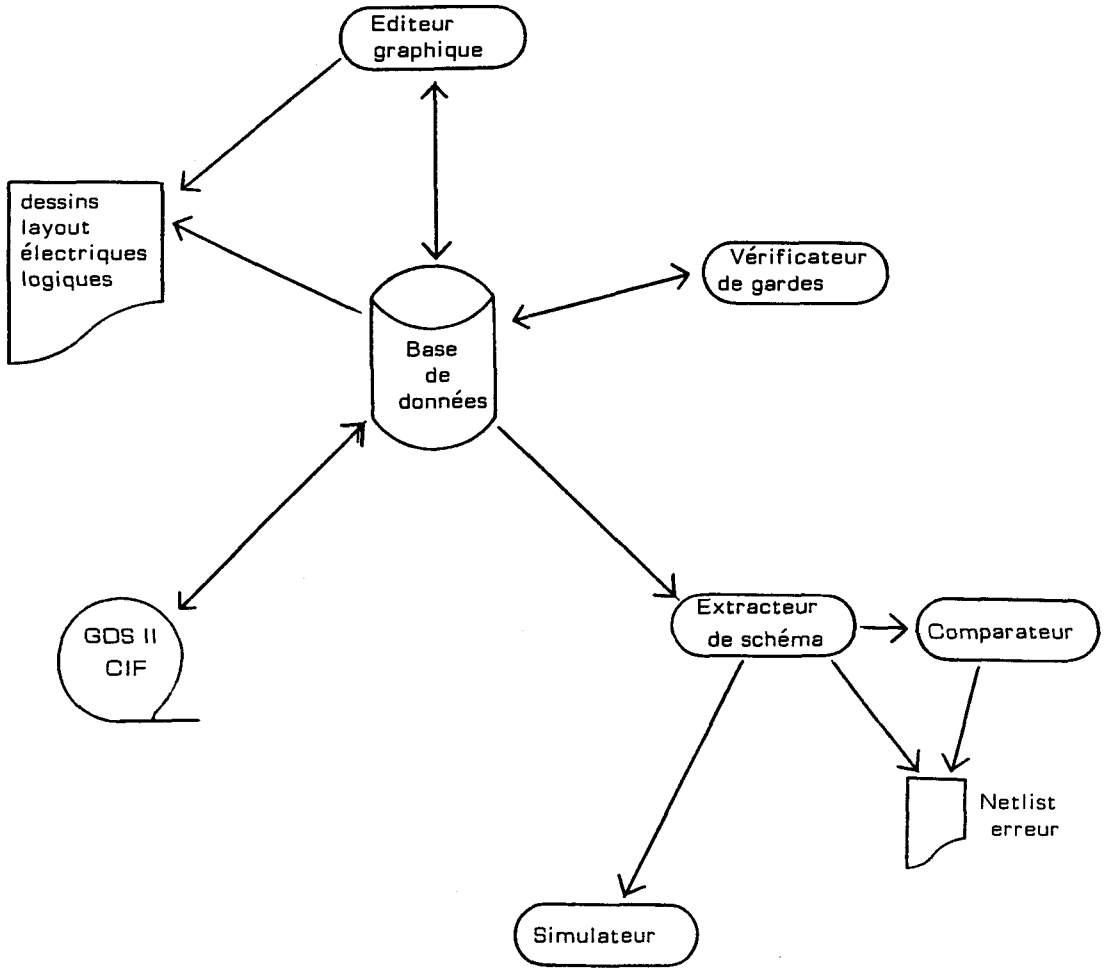
1) IMPORTANCE DE LA BASE DE DONNEES

La base de données est un élément clé de la chaîne de conception de circuit intégré. Comme nous l'avons vu, cette chaîne est constituée de nombreux logiciels. Chacun de ces produits possède sa propre structure de données et sa propre représentation des informations.

Partant de là, il est très probable que chaque maillon possèdera sa propre base de données. Ceci impose donc beaucoup de changements de formats entre ces différentes bases. Notre but était de faire une base unique commune à l'ensemble des logiciels développés. Pour ce faire, nous avons d'abord regardé les besoins de l'éditeur graphique. Nous avons ensuite vérifié que les autres produits n'utilisaient que des sous-ensembles de cette base. Une fois ce point validé, nous nous sommes attachés à la définition précise de la base de données de l'éditeur graphique, et à la définition des fonctions d'accès de haut niveau communes à tous les produits. Pour ces raisons, la base de données ne sera décrite que dans le chapitre de l'éditeur graphique, car elle est la même pour tous nos logiciels.

Cette unicité de la base de données permet donc un couplage étroit entre l'éditeur, le vérificateur des règles de garde qui extrait les données dont il a besoin directement de la base, et qui met à jour cette base en cas de détection d'erreurs. Il en est de même pour l'extracteur de schémas, en ce qui concerne la prise en compte des données. Ce produit, pour le moment, ne met pas à jour la base, car nous ne générons pas le schéma électrique sous forme graphique.

Figure 18 :



2) GENERALITES

Dans la base, ne doivent être stockées que les informations minimales permettant de reconstituer le travail réalisé par le concepteur.

- Informations générales sur chaque bloc :

  . son nom
  . son attribut
  . son échelle

- Les figures décrivant le bloc, i.e. l'ensemble des formes (2 points), leur qualificateur (type de formes : rectangles, polygones, lignes...) et le niveau de référence (encombrement, connectique, connexions (polysilicium, diffusion...)).

- Les noms des connectiques du bloc.

- La description de la descendance :

  . les blocs fils
  . leur position relative par rapport au père
  . les opérations éventuellement subies

- Toutes les informations concernant les segments se révèlent inutiles dans la base, car elles sont liées à l'entité noeud qui n'apparaît plus ici. De plus elles n'ont de signification que pour l'aspect visualisation du circuit.

- Les caractéristiques des tracés pourraient être stockées dans la base. Si ceci n'est pas réalisé, on ne peut plus modifier de façon quelconque les types de tracés, en particulier le numéro associé à un niveau. Ceci pourrait, à terme, se révéler gênant, et devrait être corrigé prochainement.

3) PROPOSITION DE STRUCTURE

Une base sera implantée sous forme d'un fichier en accès direct. Une description détaillée de la structure est donnée en annexe 3. D'une façon générale, ce fichier peut être décomposé en quatre sous-ensembles :

- L'entête de base. Premier enregistrement de la base, il contient les valeurs décrivant l'état de la base (nombre total d'enregistrements, nombre d'enregistrements utilisés, premier enregistrement de la directory et le nombre de blocs dans la base).

- La directory. Cet ensemble peut être considéré comme le catalogue de la base. Il contient l'ensemble des blocs et leurs caractéristiques générales (attribut de lecture-écriture, échelle...), et pointe vers les autres éléments constituant ce bloc.

- La description des blocs. Dans cette zone, on trouvera les enregistrements contenant les formes, les connectiques, les descriptifs et les opérations. Ces différents enregistrements sont chaînés entre eux.

- La zone libre. Cette partie est constituée d'enregistrements non occupés. Ils sont gérés sous forme de liste chaînée et/ou de liste contiguë (la liste contiguë n'a de sens que si on travaille sur un système ne disposant pas d'allocation dynamique d'espace disque tel que VM/CMS).

4) LES FONCTIONS D'ACCES EVOLUEES

Les fonctions d'accès évoluées s'appuient sur un ensemble de fonctions de plus bas niveau éventuellement appelables directement. L'interface entre ces programmes et l'application se fait par l'utilisation de buffers permettant une indépendance des fonctions d'accès à la base vis-à-vis de la structure de données de l'application. Ces programmes remplissent les fonctions suivantes :

- Obtention de la liste des bases accessibles à l'utilisateur.

- Ouverture d'une base en lecture-écriture ou en lecture seule.

- Fermeture d'une base.

- Obtention de la liste des éléments de la base. Ces éléments sont des blocs ou des circuits.

- Lecture des caractéristiques d'un bloc. Cette fonction nous retourne l'échelle du bloc, ainsi que les pointeurs permettant d'aller lire les autres entités constituant ce bloc.

- Lecture des entités du bloc à partir du pointeur fourni par la fonction précédente. Chaque élément possède son propre programme de lecture.

- Obtention d'un emplacement libre dans la base pour un type d'entité donné. Le programme appelé est différent pour un emplacement libre dans la directory de la base ou pour un emplacement dans le reste du fichier.

- Ecriture d'un enregistrement dans la directory. Cette fonction permet d'écrire un bloc dans la base. Il faut fournir à ce programme les pointeurs du bloc vers les autres entités le composant, ce qui impose d'écrire d'abord ces éléments.

- Ecriture d'une partie de bloc dans le fichier. Chaque type d'élément possède son propre programme d'écriture.

- Destruction d'un élément dans la directory de la base. Cette fonction doit être appelée après la destruction des entités constituant le bloc à détruire.

- Destruction d'une entité dans le fichier.

- Changement du nom d'un bloc.

Partant de là, il est possible de créer des fonctions d'accès beaucoup plus évoluées, mais dépendantes de la structure de données de l'éditeur graphique. Voici quelques exemples de fonctions évoluées.

- Chargement d'un bloc et de sa descendance dans la structure.

- Lecture du niveau de hiérarchie inférieure pour un bloc donné.

- Ecriture d'un bloc et des entités le composant.

- Ajout d'entités à un bloc existant.

- Suppression d'un bloc et de sa descendance.

Et bien d'autres fonctions dont l'énumération serait fastidieuse.

VII- QUELQUES FONCTIONS DISPONIBLES A L'UTILISATEUR

1) LA CONVIVIALITE

Quand on réalise un logiciel, la convivialité doit être maximale. C'est encore plus vrai pour les produits comme notre éditeur. En effet, l'utilisation du graphique doit permettre l'utilisation minimale du clavier, objet généralement peu apprécié du concepteur, une présentation plus claire de certaines fonctions (un dessin vaut souvent mieux qu'un long discours), et autoriser une gestion de menus moins hiérarchique grâce à la grande taille de l'écran.

Pour faciliter le dialogue avec l'utilisateur, l'ensemble des fonctions proposées dans notre éditeur graphique sont regroupées dans différents menus qui caractérisent un environnement de travail particulier. On distingue ainsi 3 menus chaînés :

- Le menu1, environnement d'accueil
- Le menu2, environnement de visualisation et de placement de blocs
- Le menu3, environnement d'édition d'un bloc

Pour ne pas pénaliser l'utilisateur par une structure trop hiérarchisée des menus qui le contraindrait à des changements d'environnement trop fréquents, certaines fonctions d'un environnement donné restent disponibles dans les autres.

De plus, certaines fonctions peuvent provoquer le passage dans un sous-environnement particulier. Lorsque l'utilisateur quitte ce sous-environnement, il retrouve l'état précédent : voici quelques fonctions proposées par l'éditeur, classées par environnement (le schéma des écrans est donné en annexe 4 avec la liste des fonctions).

2) L'ENVIRONNEMENT D'ACCUEIL

Cet environnement propose des fonctions :

- de manipulation des circuits de la base. Le concepteur dispose d'opérations de suppression, recopie et changement de noms de blocs ou circuits dans la base.

- utilitaires. On y trouve des fonctions permettant la communication avec l'extérieur telles que la lecture ou l'écriture de fichiers externes à la base en format GDS2 ou CIF. Les fonctions de sortie de schémas sur traceur ou fichier texte sont aussi disponibles à ce niveau. Une fonction d'aide permet au concepteur de visualiser les différentes bases qu'il possède, et leur contenu.

- de mise à jour des paramètres d'utilisation de l'éditeur, tels que la palette des couleurs et la représentation des différents niveaux.

- d'entrée en édition sur un circuit qui provoque un changement d'environnement

3) ENVIRONNEMENT DE VISUALISATION/PLACEMENT DE CIRCUIT/BLOC

Cet environnement propose des fonctions :

- de déplacement dans l'arborescence (affichage d'un bloc plein écran, changement de niveaux dans la hiérarchie...)

- de zoom sur une partie du circuit

- de manipulation de blocs. Des fonctions de déplacement et copie de blocs sont disponibles. De plus, le concepteur peut faire subir des opérations de rotation, symétrie et répétition aux différentes cellules de son circuit.

- de sortie avec ou sans sauvegarde du circuit

- d'entrée en édition sur un bloc

4) ENVIRONNEMENT D'EDITION DE BLOC

Cet environnement propose des fonctions :

- de création. Il est possible, à ce niveau, d'ajouter des rectangles, des polygones ou dess lignes appartenant à un quelconque niveau.

- de manipulation des entités. On peut déplacer, dupliquer, supprimer les éléments du bloc.

- de sortie avec ou sans sauvegarde

5) SOUS ENVIRONNEMENTS

Certaines des fonctions définies dans les menus précédents font appel à un sous environnement spécifique à ces fonctions. En voici deux exemples.

Sous menu de UPDINIT

Ce menu permet la mise à jour des couleurs et du graphisme (mise à jour de la palette et de la représentation graphique des niveaux utilisés).

Sous menu de ADD

Ce menu permet la création de formes élémentaires de niveau quelconque. L'utilisateur choisit ici le niveau et le type d'élément désiré.

VIII- CONCLUSION

Il est intéressant de comparer l'éditeur graphique réalisé avec les produits du marché sur différents points :

1) LA PORTABILITE

Notre logiciel est portable, de par l'utilisation du FORTRAN et de GKS, normes reconnues, et produits existants sur tous les calculateurs. Ceci permet donc à l'utilisateur de s'affranchir du matériel et donc du constructeur. Des portages furent réalisés sur des matériels divers (mainframes IBM, microvax, PC et PS) sans aucun problème. Ceci évite donc des investissements importants, dans la mesure où l'utilisateur dispose déjà de matériel ayant des possibilités graphiques. Qui de nos jours, ne dispose pas d'un PC avec écran graphique ?

2) LA CONVIVIALITE

Comme tous les produits existants, nous avons minimisé l'emploi du clavier et favorisé celui de la souris. Sur ce point, tous les logiciels sont égaux. Notre éditeur est cependant moins achevé sur le plan du graphisme. Par exemple, lors du déplacement d'une figure, l'utilisateur ne voit pas l'élément se déplacer avec le curseur, ce qui est le cas pour la plupart des autres logiciels. Ceci n'est pas très gênant, mais moins esthétique. Ce problème est dû à l'utilisation de GKS qui ne permet pas (dans sa version standard) de modifier l'aspect du curseur. Entre la portabilité et le raffinement graphique, nous avons opté pour une portabilité maximale.

3) LES PERFORMANCES

Sur le plan des performances, les comparaisons sont difficiles à établir. En effet, sur des stations de travail les logiciels sont nombreux. Les comparaisons ont fourni des performances équivalentes entre notre produit et les logiciels du marché. Dans la gamme PC PS, les éditeurs graphiques dans le domaine de la micro-électronique sont inexistants. Dans le domaine de la schématique logique et/ou électrique, les produits sont performants, mais non portables. Pour les cites centraux, les produits sont performants. Le seul point noir, est qu'ils exigent une grande disponibilité de la machine, ce qui impose de devoir utiliser le matériel en dessous de ses possibilités. En conséquence, pour le dessin des masques, la comparaison des performances n'a pu se faire que sur les stations de travail, et a donné de bons résultats.

Il est à noter que nombre d'autres produits ne proposent pas un affichage hiérarchique, ce qui occasionne un temps de chargement initial du circuit particulièrement long et pénible.

4) LE COUT

Le coût global se décompose en deux :

- le matériel

Celui-ci dépend de la taille des circuits à réaliser . Nos essais sur du matériel de type PC nous donnent à penser que ces machines sont limitées à des circuits de type LSI, ou à des parties de circuit plus important. Sur les autres gammes de matériel, nous n'avons pas rencontré de limites particulières.

- La formation

Sur ce point, notre solution est financièrement intéressante car elle ne nécessite que l'apprentissage d'un seul produit de par la généralisation de notre éditeur à tous les types de schématiques.

Notre produit offre donc de nombreux avantages, au détriment d'un seul inconvénient : une gestion graphique un peu sommaire. Je pense que ce problème sera résolu par les prochaines versions de la norme GKS, ou par l'utilisation d'autres normes telles que PHIGS.

LE VERIFICATEUR DE REGLES DE GARDE

Les concepts mis en oeuvre pour la réalisation d'un vérificateur de règles de gardes (c'est à dire la vérification des contraintes géométriques et électriques qui devront être respectées dans la phase de réalisation des masques) sont présentés.

Le plan de ce chapitre est le suivant :

- Les principes généraux présentant les spécificités de la vérification de règles de gardes pour les circuits intégrés.

- Les différentes méthodes existantes, présentant les concepts mis en oeuvre dans la plupart des outils existants.

- La comparaison des différentes méthodes, justifiant l'orientation des développements.

- Les fonctions disponibles, expliquant l'utilisation de notre outil.

- Le traitement des lignes obliques, présentant la généralisation des concepts de bases.

- La hiérarchisation du vérificateur, expliquant pourquoi le couplage avec l'éditeur graphique permet d'optimiser la vérification des règles de garde.

I- INTRODUCTION

Parmi les différentes phases de conception d'un circuit intégré, il en est une particulièrement longue et délicate : le dessin des masques d'implantation.

C'est en effet à ce niveau qu'interviennent les contraintes de fabrication. Ces règles, fournies par le constructeur, doivent être respectées sous peine de disfonctionnement du circuit, ou de chute des rendements à la production.

Etant donnée le nombre d'informations nécessaires à la représentation d'un circuit, on conçoit aisément qu'une vérification manuelle est impossible.

Pour cette étape de contrôle, les outils informatiques sont donc devenus indispensables.

Le travail que nous avons effectué consiste en l'étude et la réalisation d'un vérificateur de gardes (DRC) capable de travailler sur plusieurs masques.

Pour la conception de ce produit, nous nous sommes imposés les contraintes suivantes :

- faible occupation mémoire

- grande indépendance par rapport à la technologie employée

- optimisation des algorithmes pour minimiser le temps de traitement

- possibilité de portage sur micro-ordinateurs

L'un des premiers problèmes à résoudre est de trouver la représentation interne des masques à utiliser compte tenu des objectifs.

H.S. BAIRD [16] présente différentes méthodes imaginables, ainsi que les possibilités et les limitations de chacune. Chacune de ces méthodes sera reprise et explicitée ultérieurement.

Différents types de primitives graphiques élémentaires peuvent être envisagés pour la représentation des masques :

- lignes horizontales et verticales
- lignes obliques
- arcs de cercle.

En fonction du choix réalisé, le nombre de possibilités diminue fortement et se résume généralement à deux ou trois méthodes. Dans le cas qui nous intéresse -lignes horizontales, verticales et obliques- il se limite entre la représentation en bitmap, en polygones (ou contours), et en rectangles. Nous analyserons plus loin ces différentes solutions.

Une deuxième question importante est le mode de fonctionnement du programme, en d'autres termes :

- les fonctionnalités que doit remplir le programme
- la méthode de travail du concepteur qui utilisera ce programme

L'examen détaillé des systèmes existants [17] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [31] [32] [33] [34] montre que le principe de base est le même pour tous. En effet, il est démontré que pour assurer une certaine indépendance vis-à-vis de la technologie, la meilleure solution est :

- l'utilisation de fonctions de vérification travaillant sur un ou plusieurs niveaux (programmes vérifiant que les largeurs minimales et que les espacements minimaux sont bien respectés)
- l'emploi d'opérateurs logiques.

Les opérateurs logiques servent à générer des masques n'ayant pas d'existence physique sur lesquels les opérations de vérification citées plus haut seront exécutées. On montre de la même façon qu'en employant les fonctions ET, OU, XOR, MOINS et NON, on peut retrouver tous les éléments de base constituant un circuit. De cette façon, on peut vérifier pratiquement toutes les règles en générant les masques appropriés.

Pour vérifier les gardes ou les contraintes non géométriques, quelques fonctions spécifiques sont nécessaires telles que la détermination d'équipotentielles, la dilatation et la contraction de masques.

Ce mode de fonctionnement des vérificateurs de gardes a prouvé depuis longtemps qu'il est le plus efficace pour assurer l'indépendance vis-à-vis de la technologie employée, et qu'il permet de vérifier toutes les règles de gardes.

II- LES PRINCIPES GENERAUX

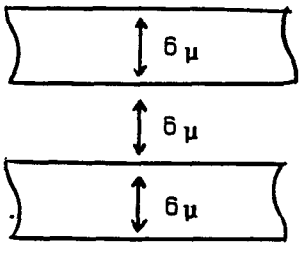
1) LES FONCTIONS DE VERIFICATION

Pour cette présentation, les masques sont constitués de rectangles. Pour les lignes obliques, les fonctions à réaliser restent les mêmes, seuls les algorithmes changent, mais les méthodes de balayage et de génération des fichiers restent les mêmes.

a. Monomasque

Il s'agit de vérifier, pour un masque, la largeur minimale des lignes et leurs espacements. Un exemple tiré d'une technologie NMOS 6 μm est donné en annexe 1. Dans ce cas, les lignes de diffusion doivent avoir au moins 6 μm de large et être espacées de 6 μm

Figure 19 :



La fonction de vérification doit être capable de vérifier ces règles, quelle que soit la configuration des lignes (parallèles, sécantes ou perpendiculaires).

- Fonctionnement du vérificateur

Il y a donc deux gardes à vérifier : la garde extérieure (espacement) et la garde intérieure (largeur).

* garde extérieure

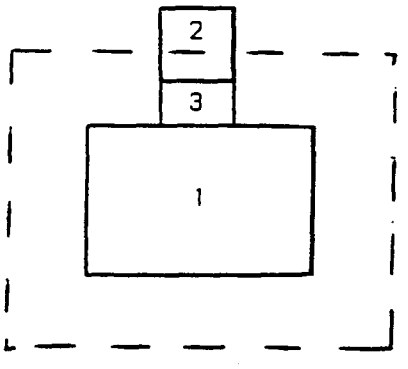
Pour chaque rectangle du masque, on effectue les traitements suivants :

- dilatation de la valeur de l'espacement
- recherche de tous les rectangles ayant une partie commune avec la figure dilatée

Plusieurs configurations sont possibles :

1) sur un côté

Figure 20 :

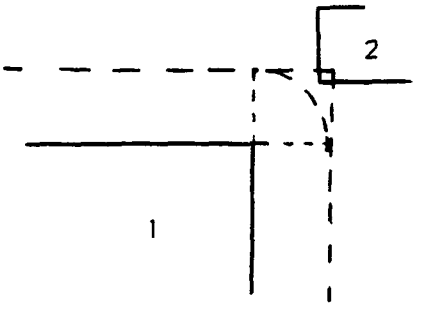


Dans ce cas, si le rectangle 3 existe, il n'y a pas d'erreur ; sinon, il y a violation de la règle.

2) sur un angle

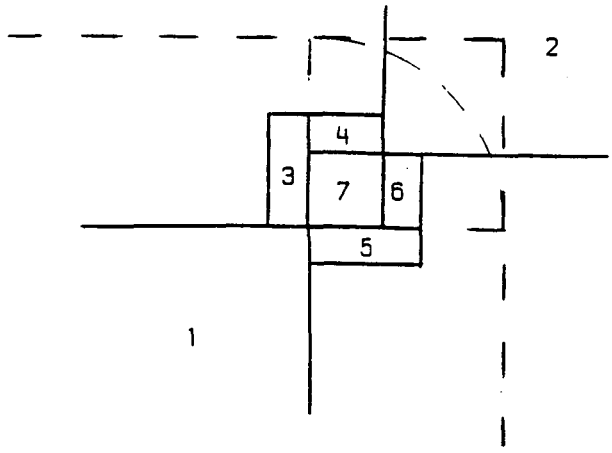
Dans ce cas, la zone de garde est un quart de cercle.

Figure 21 :



Si la partie commune à 1 et 2 se situe hors de cette zone, il n'y a pas d'erreur.

Figure 22 :



Dans ce cas, si 7 ou 3 et 4 ou 5 et 6 existent, il n'y a pas d'erreur. Si aucun n'est présent, il y a violation de la règle.

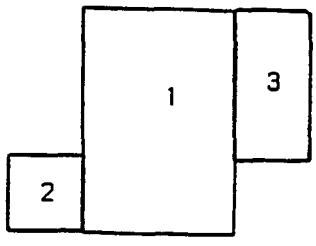
* garde intérieure

Pour chaque rectangle du masque, on effectue les traitements suivants :

. on regarde les dimensions du rectangle en X et en Y (largeur et longueur). Si celles-ci sont suffisantes (supérieure à la valeur de la garde), il n'y a pas d'erreur.

. en supposant maintenant que seule la dimension en X soit suffisante. Dans ce cas, on cherche tous les rectangles du masque ayant une frontière commune avec la figure en cours de traitement.

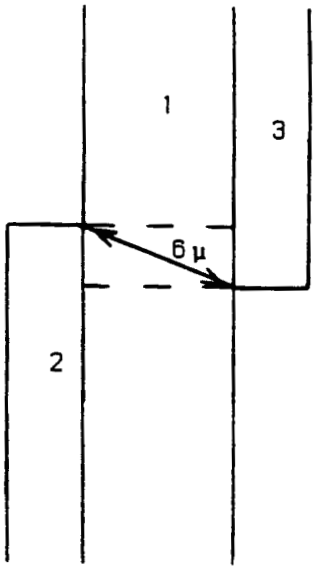
Figure 23 :



Sur la forme ainsi obtenue, on vérifie de nouveau la règle.
S'il y a encore violation, on regarde s'il n'y a pas d'autres rectangles ayant une frontière commune avec 2 (par exemple). On recommence le processus jusqu'à ce que la garde soit vérifiée ou que la forme n'évolue plus.

Il est à noter tout de même que dans la configuration suivante, les rectangles 2 et 3 ne doivent pas avoir leurs bords sur la même horizontale, car la garde intérieure doit aussi être vérifiée en diagonale.

Figure 24 :



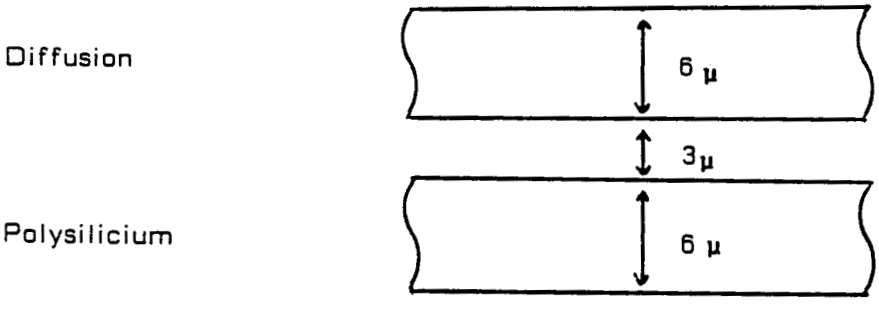
b- Multi-masques

Il existe des règles de garde faisant intervenir plusieurs masques.

* garde extérieure

exemple : en NMOS 6 μm, la diffusion et le polysilicium doivent être espacés de 3 μm.

Figure 25 :



Principe de fonctionnement

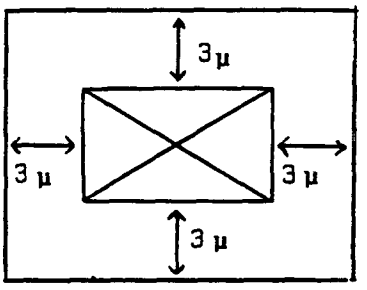
Il est semblable à la vérification de garde extérieure monomasque, excepté le fait que la recherche de tous les rectangles ayant une partie avec la figure dilatée se fait dans le second masque.

* garde de recouvrement

Un autre type de contrainte multi-masques est le débordement d'un niveau autour d'un autre.

exemple : le métal doit recouvrir les trous de contact de 3 μm.

Figure 26 :



Pour cela, on peut créer une fonction spécifique : COVER. Cette fonction n'est pas indispensable, mais permet d'éviter l'utilisation de plusieurs autres opérateurs (opérateurs logiques suivis de fonctions de vérification).

Principe de fonctionnement

Pour chaque rectangle du masque "trous de contact", on cherche dans le masque de métal, le rectangle dilaté de la valeur de la garde.

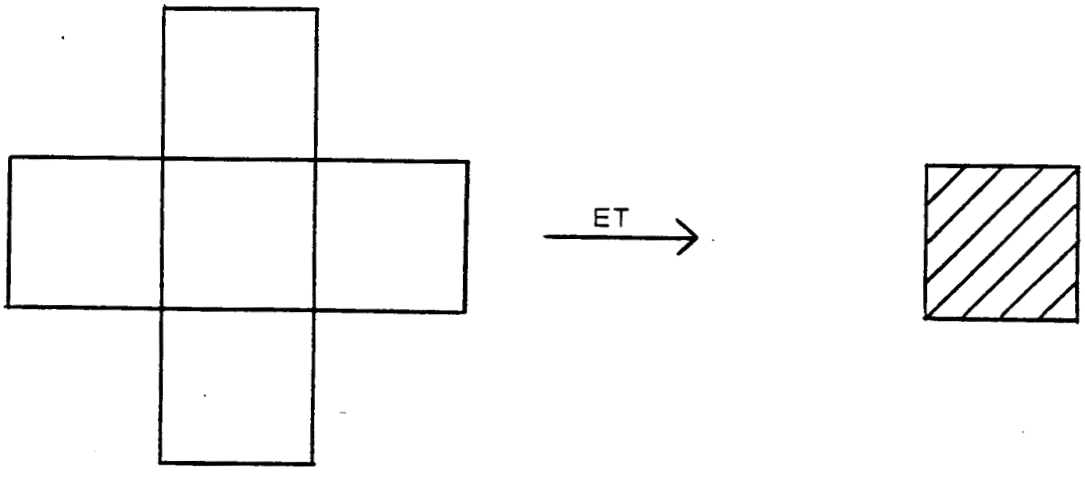
Si le rectangle existe, il n'y a pas d'erreur.
S'il n'est que partiellement recouvert par le métal, il y a une erreur.
Si la figure n'a aucune partie commune avec le métal, on regarde les options fournies par l'utilisateur, afin de savoir si ce cas doit être considère comme une violation de règle ou non.

2) LES OPERATEURS LOGIQUES

a- ET

Le ET est un opérateur qui permet de construire un masque composé de toutes les zones communes aux deux masques incidents.

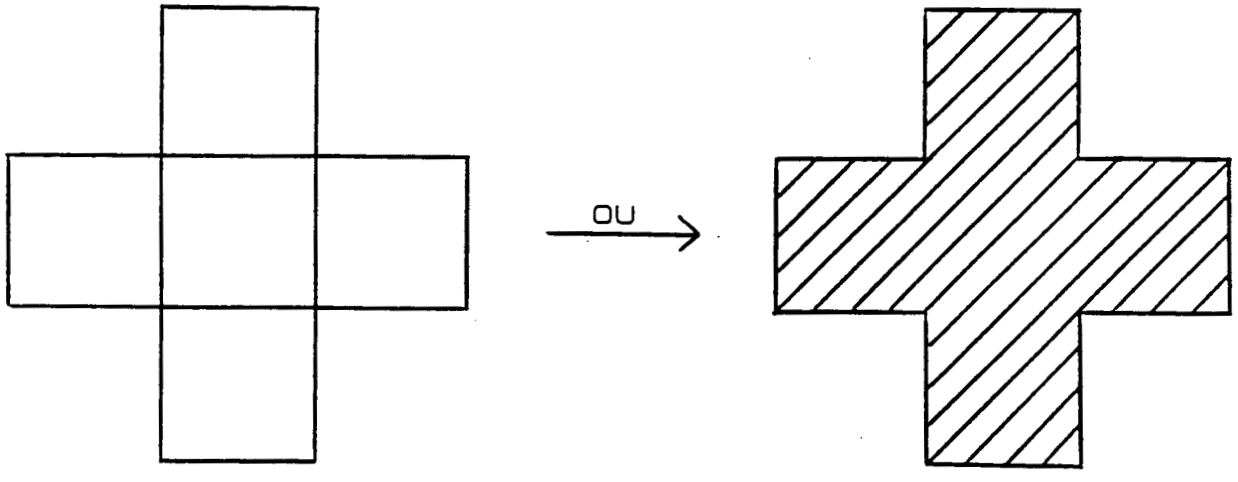
Figure 27 :



b- OU

Le OU est un opérateur qui permet de construire un masque construit par fusion de deux masques incidents.

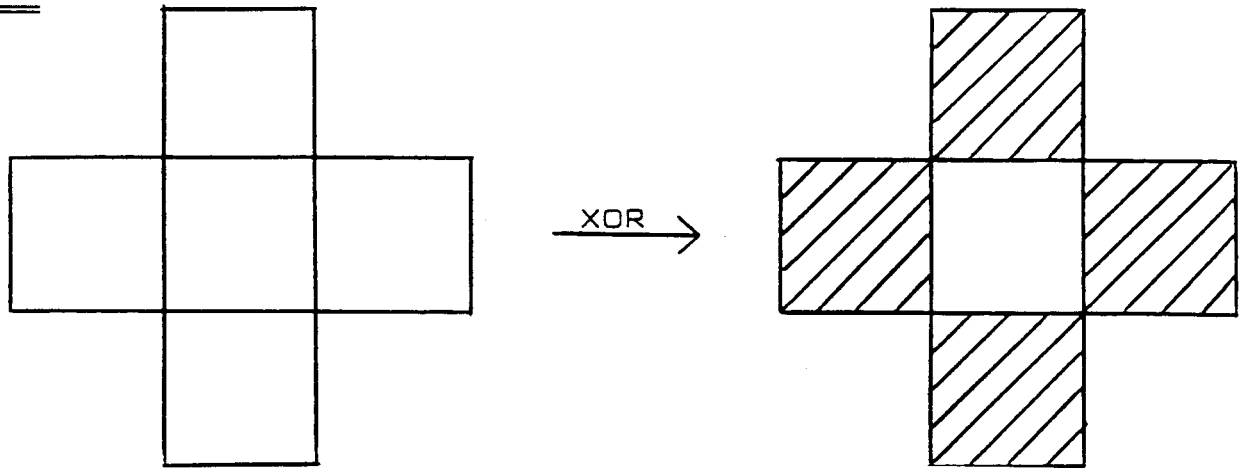
Figure 28 :



c- XOR

Le XOR, tout comme le OU, est un opérateur qui permet de construire un masque construit par fusion de deux masques incidents à l'exclusion des zones communes.

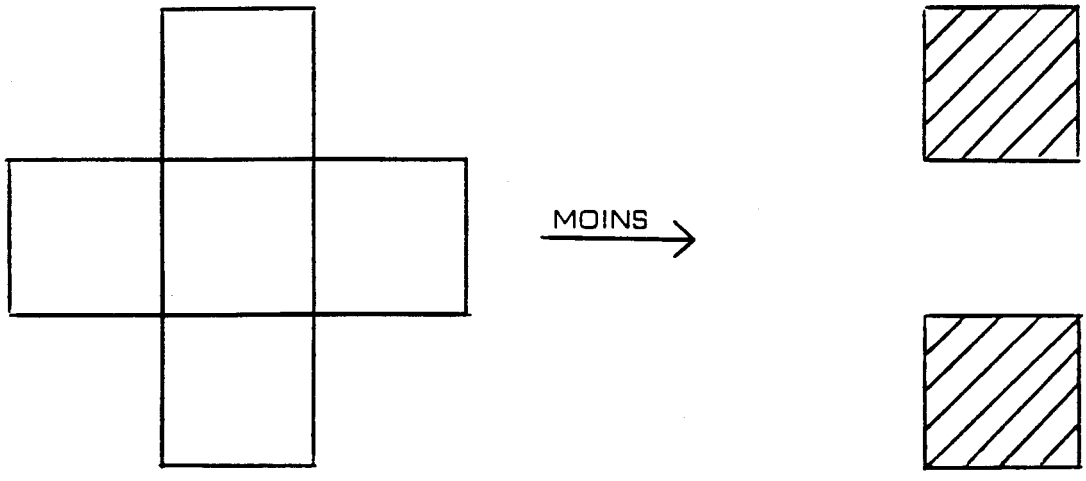
Figure 29 :



d- MOINS

Le MOINS est un opérateur qui permet de construire un masque identique au masque incident auquel on a retiré toutes les parties communes avec le deuxième masque. Le MOINS n'est pas un opérateur logique, mais est le regroupement de deux fonctions : un ET suivi d'un XOR.

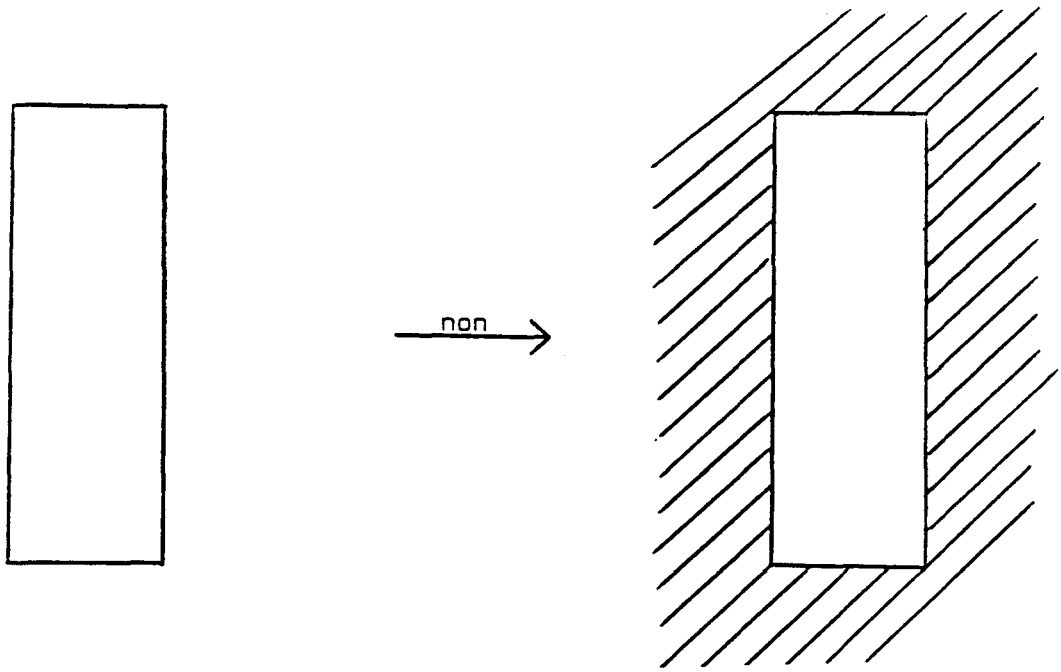
Figure 30 :



e- NON

Le NON est un opérateur qui permet de construire un masque inverse du masque incident

Figure 31 :



3) LES FONCTIONS PARTICULIERES

D'autres fonctions peuvent être utiles. Par exemple, une fonction de dilatation d'un masque. Cet opérateur peut servir à générer les surtrous autour d'un trou (on appelle surtrou une zone d'oxyde mince entourant un trou de contact ayant pour but d'éviter une rupture de métal). A ce niveau, tout est envisageable, mais ces fonctions n'influent pas ou peu sur le choix de la représentation interne. Bien souvent, ces opérateurs ne sont que le regroupement d'une suite d'opérations plus simples, dans le seul but de gagner du temps en évitant de nombreuses lectures et écritures.

III LES DIFFERENTES METHODES EXISTANTES

Les différentes solutions qui peuvent être envisagées pour la représentation d'un masque sont successivement analysées et discutées dans ce paragraphe :

- bit-map
- segments
- rectangles

1) "BIT-MAP"

a- Généralités

On découpe la surface du circuit en un ensemble de carrés élémentaires de côté unitaire. La valeur de l'unité dépend de la technologie employée. Par exemple, pour notre technologie NMOS 6 μm, l'unité de base sera le μm.

On suppose dans cette étude que les dimensions maximales du circuit sont 50 000 par 50 000 unités (fonction de la technologie).

Chaque rectangle élémentaire est codé sur deux bits (deux bits sont nécessaires pour réaliser les fonctions travaillant sur deux ou plusieurs masques).

Pour un masque, on aura donc 5 000 000 000 bits soit 596 M octets.

La mémoire centrale à notre disposition étant de l'ordre de 4 M octets, il faudra donc découper le dessin en 149 sous-dessins.

Cette segmentation pose le problème de recouvrement entre les différents sous-dessins afin de pouvoir vérifier les gardes sur chaque sous-ensemble séparément. Le nombre de découpes sera donc augmenté.

Le chiffre de 596 M octets justifie à lui seul l'abandon de cette solution. En effet, un circuit étant généralement constitué d'une dizaine de masques, l'espace disque nécessaire est prohibitif, même pour de gros systèmes.

b- Vérification de gardes

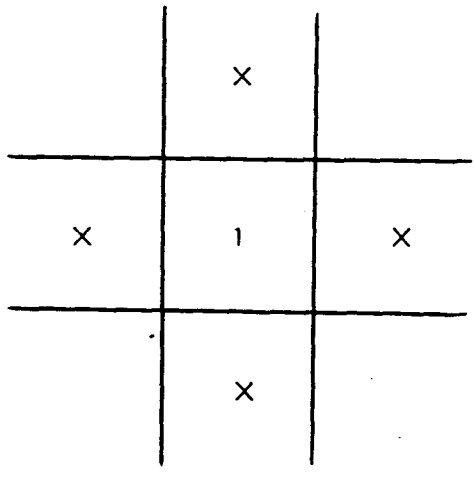
1) remplissage

Pour chaque carré élémentaire de la bit-map, on regarde s'il appartient au masque. Si oui, on met le bit correspondant à 1. Si non, on le met à 0.

2) fonctionnement

Pour la vérification de garde extérieure, on balaie le sous-dessin point par point. Si on trouve 0, on passe au point suivant. Si on trouve 1, on teste les quatre points les plus proches.

Figure 32 :



On distingue trois types de points :

- point à l'intérieur. On ne fait aucun traitement

- point sur le bord. On vérifie les points au voisinage du point origine. La direction est donnée par l'extérieur de la figure

- point sur un coin. On vérifie les carrés appartenant au quart de cercle.

Pour la garde intérieure, le traitement est le même sous réserve d'échanger le rôle des 0 et des 1.

c- Fonctions logiques

On remet la Bit-Map à zéro. Pour un opérateur à deux opérandes, on prend le premier masque et on met 1 dans les carrés élémentaires contenus dans le niveau en cours de traitement.

On prend ensuite le deuxième masque et on ajoute 1 dans les carrés inclus dans ce deuxième niveau.

Pour le résultat, on sélectionnera les figures élémentaires en fonction de l'opération réalisée.

Pour l'opérateur ET, on prendra l'ensemble des carrés contenant 2.

Pour le OU, la valeur doit être 1 ou 2.

Pour le XOR, on ne tient compte que des figures contenant 1.

Pour le NON (opérateur à un seul opérande), on ne tient compte que des 0.

Pour la fonction MOINS, le remplissage de la Bit-Map ne se fait pas de la même façon. Lors de la prise en compte du deuxième masque, on met 0 dans tous les carrés inclus dans ce deuxième niveau.
Ensuite, on ne prend en compte que la valeur 1.

Quelle que soit l'opération à réaliser, il faut donc balayer toutes les figures élémentaires afin d'obtenir le résultat.

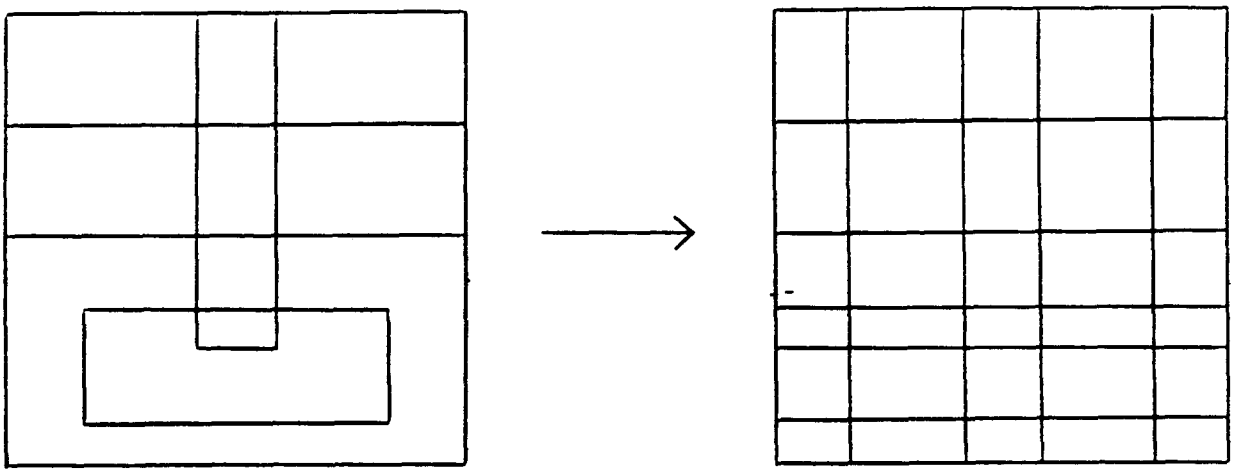
d- Variantes

Deux variantes de cette méthode ont été envisagées.

1) Grain variable

Dans cette configuration, le côté des rectangles élémentaires n'est pas constant. Il prend pour valeur la distance entre deux segments consécutifs.

Figure 33 :



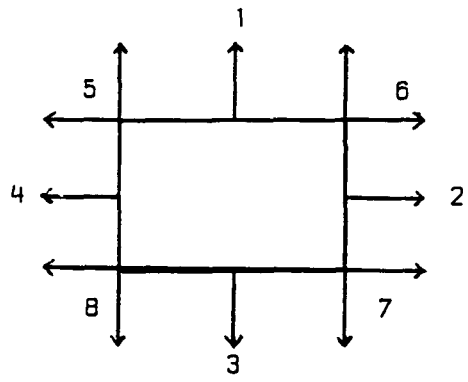
Dans ce cas, pour une technologie NMOS 6 μm, on obtient un gain de place d'un facteur 5. La place nécessaire reste donc très importante.

De plus, le balayage de cette Bit-Map devient complexe, et par conséquent, coûteux en temps CPU.

2) Contours en Bit-Map

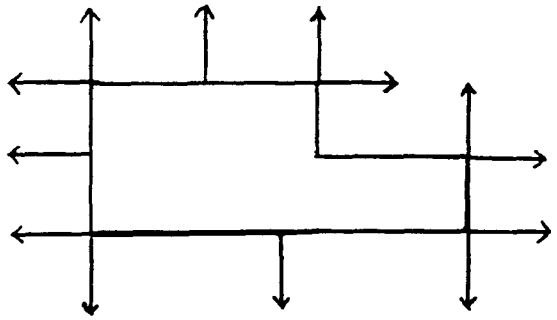
Dans cette configuration, seuls les contours sont représentés. L'avantage majeur est la rapidité car seules les frontières sont testées. Par contre, il faut beaucoup plus d'informations par point (4 bits). En effet, au bit indiquant si on est sur un contour ou non, il faut ajouter une notion d'extérieur. Huit cas existent, ce qui nécessite 3 bits supplémentaires.

Figure 34 :



De plus, à chaque bit de contour, il faut vérifier s'il n'est pas déjà à 1. Si oui, la figure est jointe à une autre. Il faut remettre à 0 la partie commune exceptés le premier et le dernier bit du segment. De plus, il faut remettre les bons indicateurs d'extérieur.

Figure 35 :



A la fin de cette étape, les figures sont faites.

2) "SEGMENTS"

a- Généralités

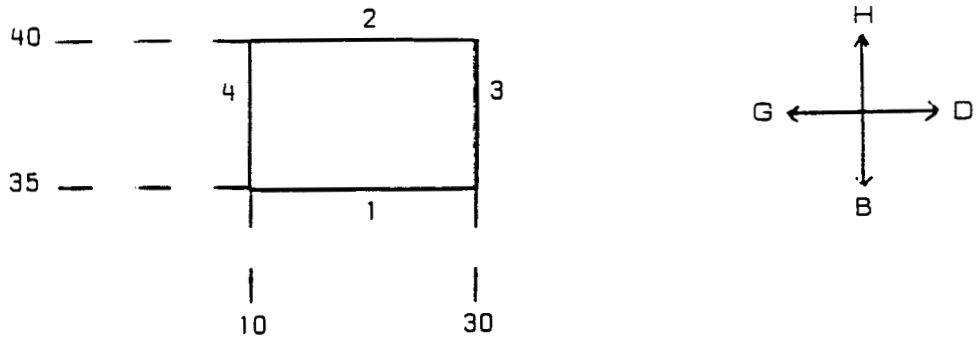
On découpe chaque rectangle en quatre segments, deux segments horizontaux et deux segments verticaux.

Chaque segment sera défini par :

- abscisse (ou ordonnée)	4 octets
- ordonnée début (ou abscisse début)	4 octets
- ordonnée fin (ou abscisse fin)	4 octets
- extérieur (bas, haut, droite, gauche)	4 octets
- angle (obtus = O, aiguë = A)	
	16 octets

Pour les opérateurs logiques, on ajoutera à ces segments deux liens de chaînage pour accélérer le traitement, ce qui portera la place occupée à 24 octets par segment.

Figure 36 :



Ce rectangle sera représenté par les quatre segments suivants :

1	35	10	30	B00	)	
					)	horizontaux
2	40	10	30	H00	)	
3	10	35	40	G00	)	
					)	verticaux
4	30	35	40	D00	)	

Les angles sont codés de la façon suivante :

Figure 37 :

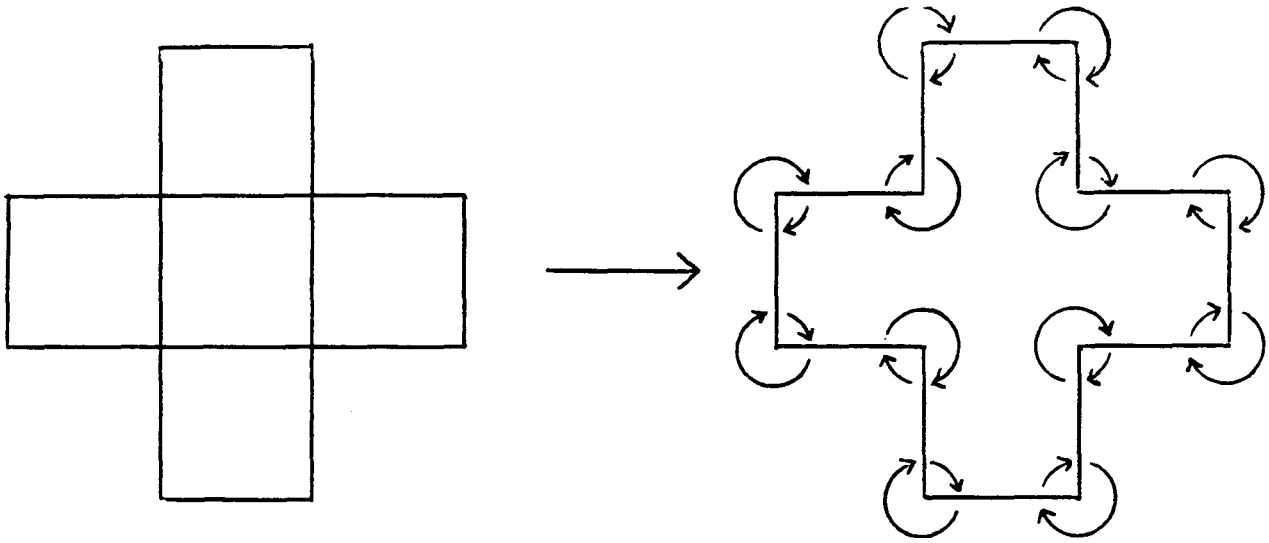
Figure 38 :

On génère ainsi deux fichiers pour chaque masque ; un pour les segments horizontaux, un autre pour les verticaux. Ils sont triés suivant les critères abscisses et ordonnées de début (ou ordonnées et abscisses de début).

1) Création des figures

On veut supprimer les segments ou morceaux de segments se trouvant à l'intérieur d'une figure.

Figure 39 :

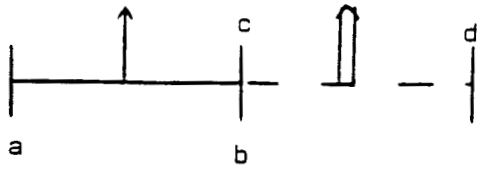


Aux formes résultantes, on ajoutera les liens de chaînage nécessaires aux opérateurs logiques.

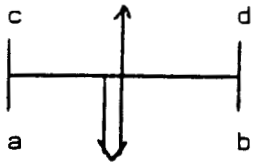
Pour réaliser ces opérations, on traite d'abord le fichier des segments horizontaux.

Les cas suivants doivent être envisagés :

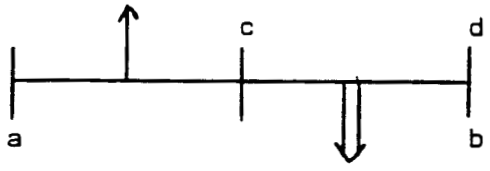
Figure 40 :



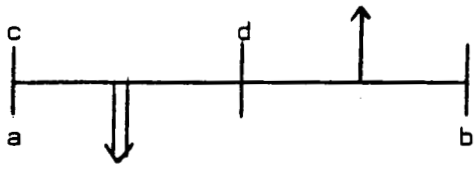
Création du segment ad



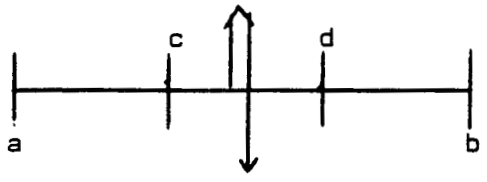
Suppression des 2 segments



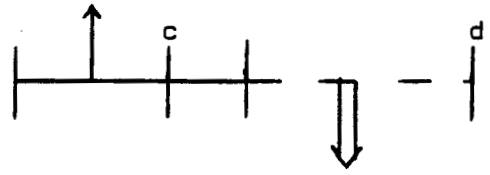
Création du segment ac
Suppression de ab et cd



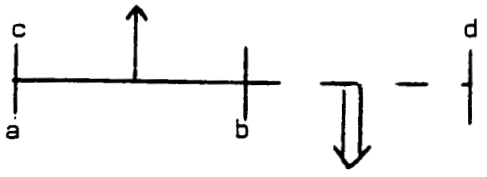
Création du segment db
Suppression de ab et cd



Création des segments ac et db
Suppression de ab et cd



Création des segments ac et bd
Suppression de ab et cd



Création du segment bd
Suppression de ab et cd

Il faut également rectifier la notion d'extérieur et d'angle.

Pour les segments verticaux, on effectue le même traitement sans rectifier les notions d'extérieur et d'angle.

b- Vérification de gardes

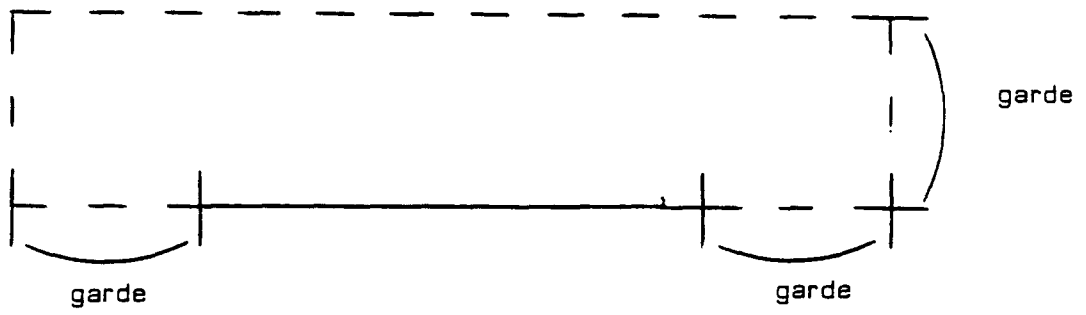
1) Garde extérieure

a- segments horizontaux

Pour chaque segment de ce fichier, on effectue les traitements suivants :

- génération d'un "Rectangle" de garde.

Figure 41 :



- on cherche tous les segments du fichier ayant une intersection avec cette figure. On détermine ainsi les erreurs. Il faut noter cependant que la garde sur les coins est un quart de cercle.

b- segments verticaux

La méthode est identique à celle utilisée pour les segments horizontaux. Pour éviter la redondance, on ne vérifie pas les quarts de cercle sur les coins.

2) Garde intérieure

On utilise le même algorithme que pour la vérification de garde extérieure en changeant les notions d'angles et d'extérieur.

Pour chaque segment, on aura donc les transformations suivantes :

Aiguë	->	Obtus
Obtus	->	Aiguë
Haut	->	Bas
Bas	->	Haut
Gauche	->	Droite
Droite	->	Gauche

c- Fonctions logiques

On part des fichiers de segments sur lesquels on a effectué la création des formes.

On prend un polygone de chaque masque (pour un opérateur à deux opérandes) et on cherche les intersections des segments de ces deux figures.

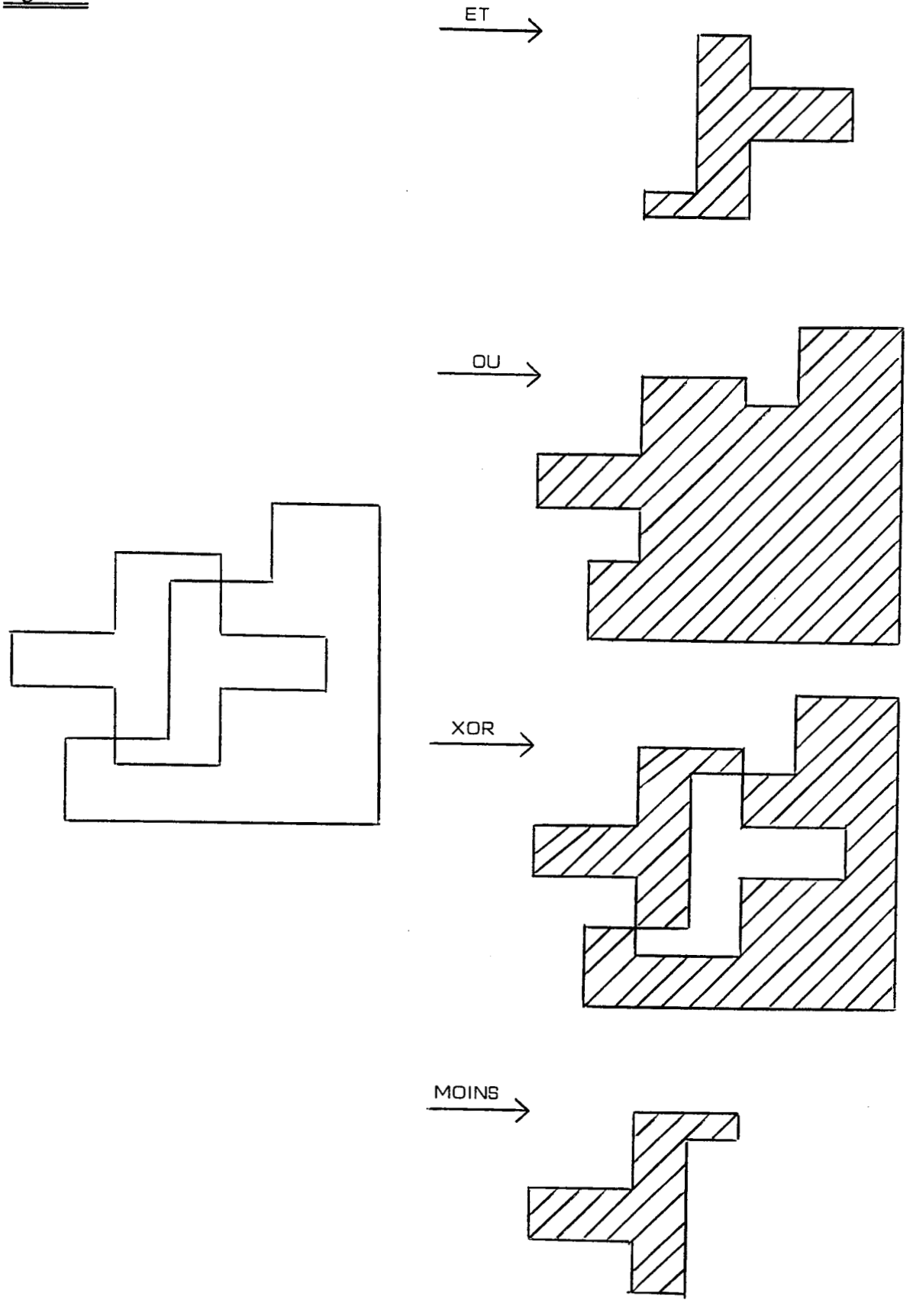
Les morceaux de droites compris entre deux points sont à traiter pour obtenir le résultat.

Pour le ET, le masque final est constitué de l'ensemble de ces segments.

Pour le OU, il faut supprimer ces traits.

Pour le XOR et le MOINS, il faut modifier les notions d'angle et d'extérieur de ces morceaux de droites.

Figure 42 :



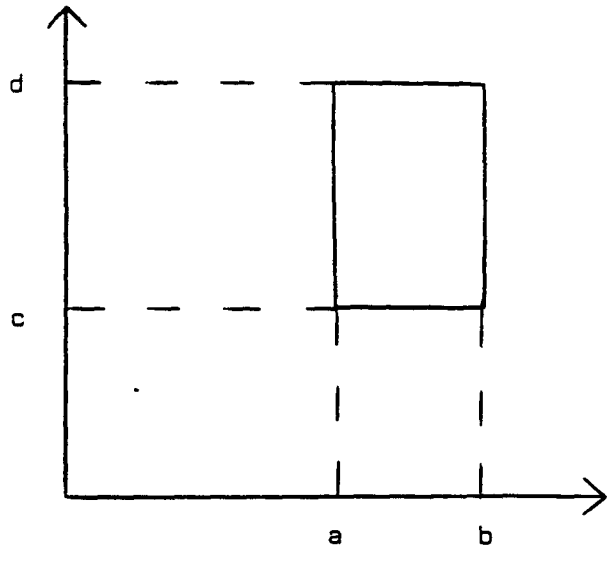
Ces algorithmes sont complexes à gérer et nécessitent de calculer beaucoup d'intersections de segments, problème complexe et coûteux en temps d'exécution dans le cas de segments quelconques.

3) "RECTANGLES"

a- Généralités

Un rectangle est codé sur quatre entiers qui sont les projections des coins sur les axes X et Y.

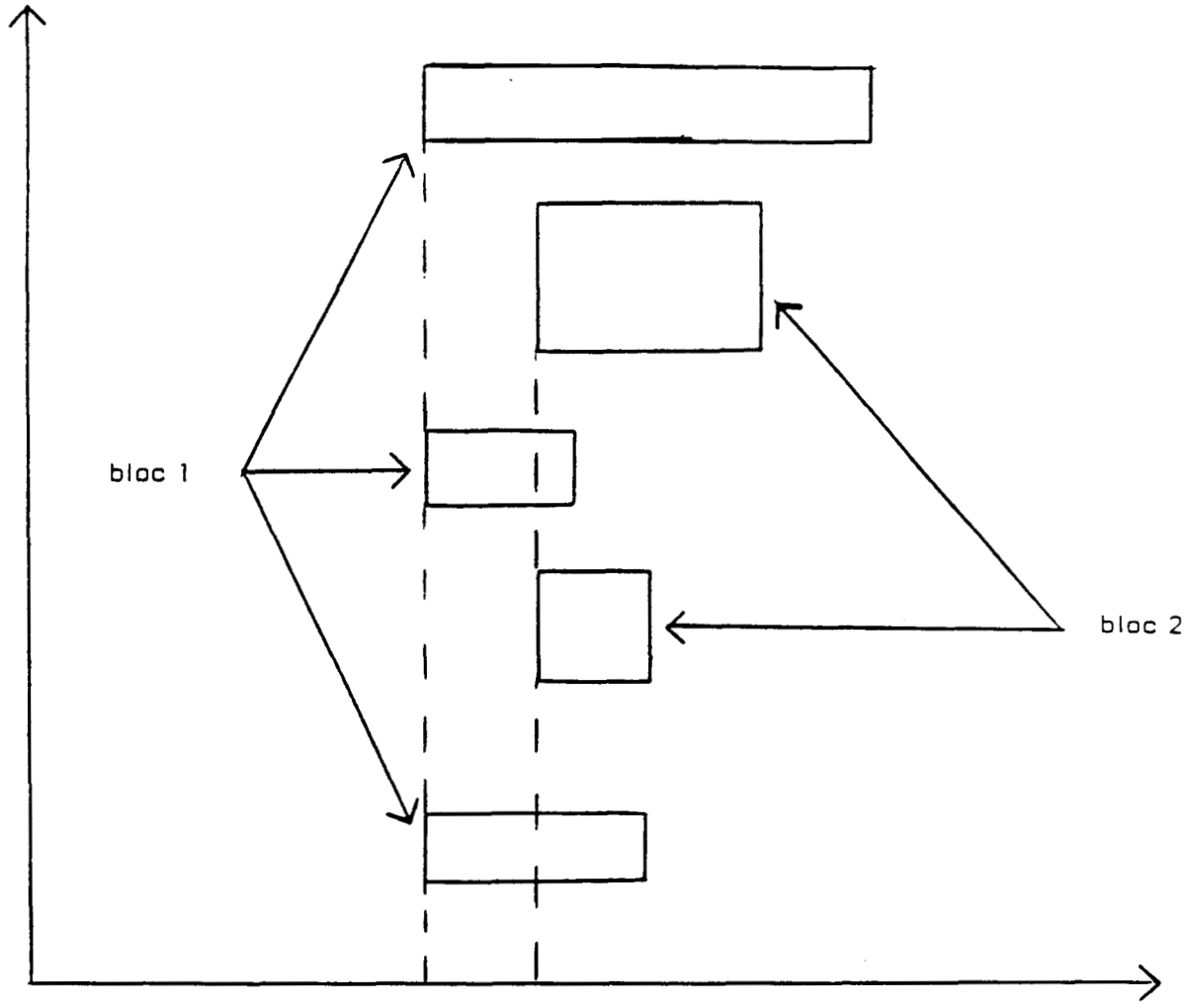
Figure 43 :



On génère ainsi un fichier par masque. Il sont triés en abscisses et ordonnées croissantes. On obtient des blocs de rectangles à valeur de a (valeur minimale de la projection sur l'axe X des coins du rectangles) égale. Aux quatre entiers codant le rectangle, on ajoute un pointeur afin d'accélérer le balayage (cf schéma).

Un enregistrement est constitué de 20 octets et il n'y a qu'un fichier par masque. On obtient ainsi un gain de place de plus de quatre par rapport au codage en segments.

Figure 44 :



b- Fonctions de vérifications

Le principe de fonctionnement de ces opérateurs de vérifications, pour la représentation en rectangles, a été vu lors de la présentation des fonctions.

c- Fonctions logiques

Après analyse des opérateurs logiques, il apparait que, pour accélérer le traitement, il faut connaître toute la surface de la figure en un seul enregistrement. C'est pour cette raison que la représentation en rectangles a été étudiée.

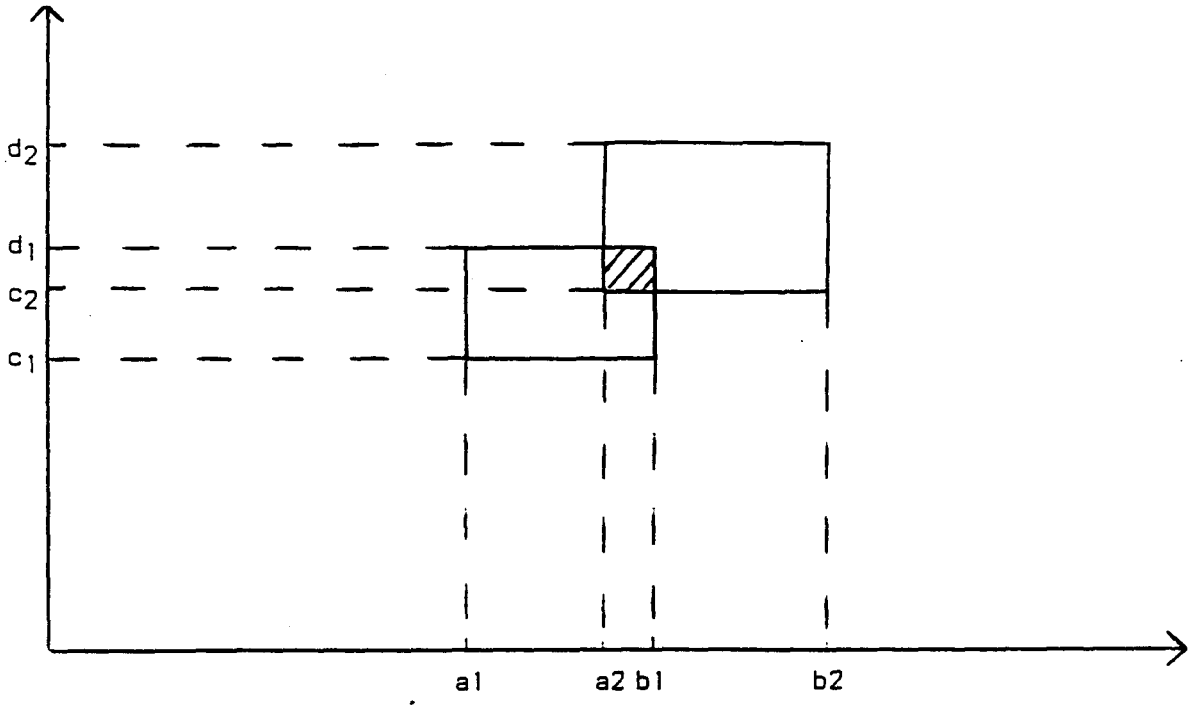
Exemple : l'opérateur ET

En partant de deux rectangles ayant une partie commune, on montre que l'intersection a pour coordonnées :

max (a1,a2), min (b1,b2), max (c1,c2), min (d1,d2)

où (a1,b1,c1,d1) et (a2,b2,c2,d2) sont les valeurs de projection des rectangles sur les axes.

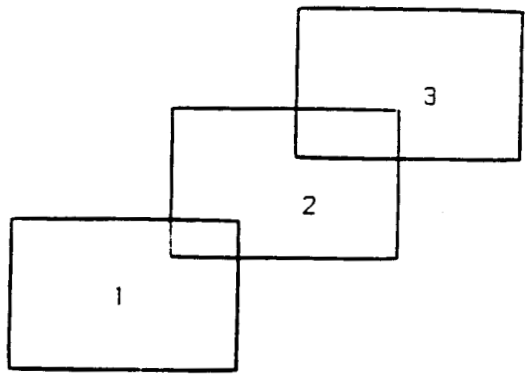
Figure 45 :



Pour les fonctions OU, XOR, MOINS, et NON, les opérations à effectuer sont similaires. Cependant, ces opérations imposent de faire des insertions et suppressions dans le fichier ce qui justifie pleinement le lien de chaînage mis en place dans le codage, ainsi que le tri et les critères de tri.

Le traitement des trois rectangles suivants montre clairement le problème.

Figure 46 :

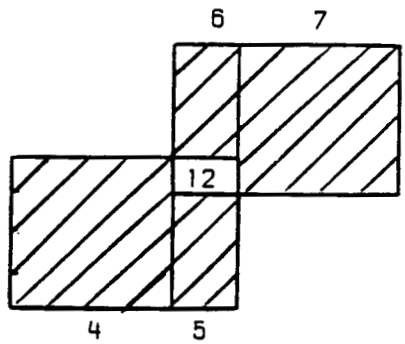


1 et 3 appartiennent au premier masque, 2 au second.

On se propose de réaliser la fonction XOR.

On traite les rectangles 2 à 2. Dans le cas qui nous intéresse, on traitera d'abord les rectangles 1 et 2.

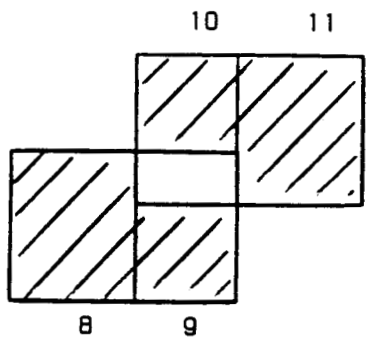
Figure 47 :



Le résultat est l'ensemble des quatre rectangles hachurés.

On répète l'opération avec 2 et 3

Figure 48 :



Si on considère que le résultat est composé des rectangles 4 à 11, on s'aperçoit que 8 recouvre la partie 12 qui ne doit plus exister.

Pour éviter ce problème, il faut faire le XOR entre les rectangles 7 et 3 au lieu de 2 et 3. Ceci nécessite de supprimer 2 et d'insérer 6 et 7.

Il faut donc choisir une structure de données adaptée à de telles opérations.

IV- COMPARAISON DES DIFFERENTES METHODES

Sur le plan de la place mémoire, la représentation en rectangles est de loin la plus économique. Par conséquent, le nombre de découpes éventuelles en sous-dessins sera très limité, ce qui occasionne un gain important au niveau du temps d'exécution. Pour ces raisons, la méthode Bit-Map ne peut pas être retenue. De plus, cette méthode impose un test de l'espace vide, ce qui représente une perte de temps assez importante.

Pour les possibilités restantes (segments et rectangles), un tri des fichiers sera nécessaire. Là encore, les rectangles sont plus avantageux, car la quantité à trier est plus petite.

Pour la détermination des figures (équipotentielles), la méthode segments est plus avantageuse, puisque, par construction, les figures existent. Par contre, la représentation rectangles ne permet pas facilement de les déterminer. Cela impose un balayage des masques pour trouver les éléments tangents entre eux. Cette détermination des figures est quasi inutile pour la vérification de gardes, et n'est donc pas pénalisante.

Pour la programmation des fonctions, la complexité est identique dans les deux cas. Pour les segments, les fonctions de vérification sont plus simples à programmer que les opérateurs logiques. C'est l'inverse pour les rectangles.

Pour les deux représentations, toutes les erreurs sont détectables.

Pour le temps d'exécution, on constate, en analysant les algorithmes, que :

- les vérificateurs prennent sensiblement le même temps pour les deux représentations. Ceci est vrai dans la mesure où le concepteur n'a pas génère énormement de petits rectangles. C'est pourquoi une fonction de compactage du schéma est à envisager, qui a le même rôle que la fonction de génération des formes dans la représentation en segments.

- les opérateurs logiques seront plus rapides en rectangles qu'en segments car il n'y a pas de calcul d'intersections de segments à effectuer. Le gain sera de plus de deux.

Pour ces raisons, nous avons choisi la représentation en rectangles pour la réalisation de notre logiciel.

V- LES FONCTIONS DISPONIBLES

Un exemple d'encodage complet d'une technologie NMOS 6 μm est donné en annexe 1.

1) LA VERIFICATION D'UN MASQUE

La fonction de vérification mono-masque est la fonction VERIF. Cet opérateur réalise à la fois la vérification de garde intérieure et extérieure. La syntaxe est la suivante :

VERIF masque garde-intérieure garde-extérieure

Si l'une des valeurs de garde est nulle, la vérification correspondante n'est pas effectuée.

2) LA VERIFICATION DE PLUSIEURS MASQUES

Plusieurs fonctions de vérification multi-niveaux sont disponibles :

a- Garde extérieure multi-masques

SPACING fait la vérification de garde extérieure entre deux masques différents. Cet opérateur considère que les recouvrements entre les masques sont autorisés. A la demande de l'utilisateur, il autorise ou non les tangences couvertes par un troisième niveau. La syntaxe est :

SPACING masque 1 masque 2 garde extérieure [(options]

Il est possible de mettre des conditions de vérification telles que :

TOUCH CASE masque 3

Ceci autorise la tangence entre les deux niveaux si le niveau 3 est présent à l'endroit de la tangence.

b- Garde de recouvrement de masques

COVER fait la vérification de recouvrement d'un masque autour d'un autre. A la demande de l'utilisateur, cet opérateur autorise ou non l'absence de recouvrement. Un recouvrement partiel est toujours une erreur. La syntaxe est :

masque 1 **COVER** masque 2 [(**ALL**]

L'option ALL impose un recouvrement de tous les éléments du niveau 2.

3) LES FONCTIONS LOGIQUES

a- ET

La fonction ET génère un masque résultant qui est l'intersection de deux masques. La syntaxe est la suivante :

masque = masque 1 **ET** masque 2

b- OU

La fonction OU génère un masque résultant qui est la fusion de deux masques. La syntaxe est la suivante :

masque = masque 1 **OU** masque 2

c- XOR

La fonction XOR génère un masque résultant qui est la fusion de deux masques auquel on enlève l'intersection des deux niveaux de départ. La syntaxe est la suivante :

masque = masque 1 **XOR** masque 2

d- MOINS

La fonction MOINS génère un masque résultant qui est le masque 1 auquel on a enlevé l'intersection des deux niveaux d'origine. La syntaxe est la suivante :

masque = masque 1 **MOINS** masque 2

e- NON

La fonction NON génère un masque qui est l'inverse du masque origine. La syntaxe est la suivante :

masque = **NON** masque 1

4) LES FONCTIONS PARTICULIERES

- La fonction PLUS permet de dilater un masque. La syntaxe est la suivante :

masque = masque 1 **PLUS** valeur

- La fonction ECRIT permet de générer un fichier sous forme de texte contenant un masque. Ceci permet par exemple de générer le niveau "surtrou" à partir du niveau trou de contact et de la fonction PLUS. La syntaxe est la suivante :

ECRIT masque 1

- La fonction GROUP permet de compacter un niveau. Cet opérateur remplace un ensemble de petits rectangles en un seul de taille supérieure (quand cela est possible). Ceci permet d'accélérer les traitements ultérieurs de ce niveau. La syntaxe est :

GROUP masque 1

- La fonction EQ permet de vérifier que les éléments d'un niveau ont une taille précise. Cet opérateur est utilisé, par exemple, pour vérifier que les trous de contact ont les bonnes dimensions. La syntaxe est la suivante :

EQ masque val1 val2 val3 val4

L'utilisateur peut fournir deux paires de valeurs représentant les dimensions autorisées pour les rectangles.

- La fonction SURFM permet de vérifier que les équipotentielles d'un niveau ont une surface minimale (par exemple les zones actives en technologie CMOS). La syntaxe est la suivante :

SURFM masque surface

- La fonction ERROR permet d'écrire un masque dans le fichier résultat dont tous les éléments sont en erreur. La syntaxe est la suivante :

ERROR masque

- La fonction MEET génère un masque dont les éléments sont les rectangles des niveaux d'origine ayant une intersection non nulle. La syntaxe est la suivante :

masque = **MEET** masque 1 masque 2

- La fonction VERITR vérifie les débordements de diffusion et de polysilicium autour des transistors. Cette fonction a été créée pour des raisons de performances. La syntaxe est la suivante :

VERITR masque-transistor masque-polysilicium masque-diffusion valeur-polysilicium valeur-diffusion

- La fonction DIRECT vérifie une garde de recouvrement d'un masque sur un autre. La différence avec COVER réside dans le fait que ce recouvrement peut être asymétrique. La syntaxe est la suivante :

DIRECT masque 1 masque 2 val1 val2

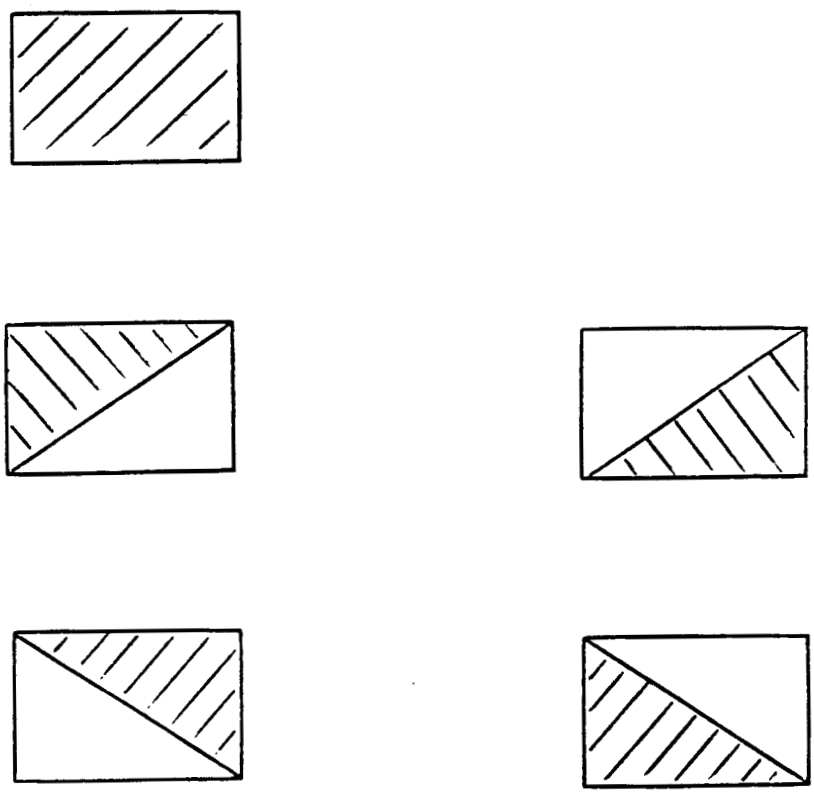
VI- LES LIGNES OBLIQUES

1) POSITION DU PROBLEME

Le codage utilisé ne permet pas, sans modification, le traitement des lignes obliques. Avec l'avènement de technologies toujours plus fines, cette limitation est gênante. Pour résoudre ce problème, plusieurs étapes sont à franchir.

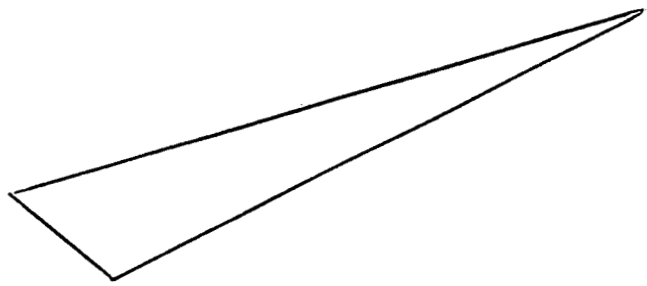
Il faut d'abord trouver un moyen de coder les obliques dans la structure existante. La seule entité existante étant le rectangle, la seule solution envisageable est d'utiliser une partie de ces rectangles. Nous obtenons donc deux types de figures, les rectangles et les triangles-rectangles.

Figure 49 :



Il nous suffit donc d'ajouter à la structure une information précisant si la figure est un rectangle ou un triangle, et dans ce cas, quel est l'orientation de ce triangle-rectangle. Se pose maintenant le problème de la découpe des masques. En effet, certaines formes ne peuvent pas être transformées en un ensemble de rectangles et triangles-rectangles.

Figure 50 :



Ceci n'est pas gênant dans la mesure où ces formes présentent forcément des erreurs de garde intérieure. Nous pouvons donc les éliminer avant un quelconque traitement. Nous pouvons donc coder les obliques dans notre structure.

2) LA VERIFICATION DE GARDE

Pour la vérification des gardes intérieures et extérieures, cette modification ne change pas le principe de base du traitement. Le parcours de la structure et la recherche de tangence de rectangles ne changent pas. Les différences résident dans l'existence éventuelle de la tangence pour un triangle, et dans le calcul des distances. Le traitement n'est pas modifié, il est simplement complété.

3) LES OPERATEURS LOGIQUES

Pour la réalisation des opérations logiques, le problème est complexe. En effet, pour les opérateurs rectangles tels que le XOR, on traite tous les cas d'intersections de deux rectangles de façon particulière. Ceci est possible, car les configurations différentes sont au nombre de 16. En ajoutant les triangles-rectangles, ce nombre est passé de 16 à plus d'un millier. Il est donc impossible de conserver ce mode de fonctionnement. Pour résoudre ce problème, la seule solution envisageable est la suivante :

- Pour les intersections de rectangles, on conserve ce principe.
- Pour les intersections mettant en oeuvre un ou deux triangles, on revient à une description des figures sous formes de segments. A partir de là, on calcule la ou les formes résultantes que l'on découpe à nouveau sous forme de rectangles et triangles-rectangles.

Cette solution ne met pas en cause le codage rectangle, car la majorité des figures constituant un masque sont des rectangles. On conserve donc les performances de la structure initiale. La détermination des intersections des segments n'est pas trop pénalisante dans la mesure où le nombre maximum de segments à traiter à un instant donné est de sept, ce qui limite fortement les possibilités.

Un autre point à considérer est la découpe des figures résultantes. A ce niveau, il est possible d'obtenir des figures non "découpables". Dans le cas d'une vérification de règles de garde, ceci n'est pas gênant. En effet, si un masque est en erreur, tous les niveaux résultants de combinaisons avec ce masque le seront aussi. Pour les autres cas, l'utilisateur a la possibilité de demander au programme une approximation de la forme avec la précision qu'il désire.

Ceci n'est pas gênant dans la mesure où ces formes présentent forcément des erreurs de garde intérieure. Nous pouvons donc les éliminer avant un quelconque traitement. Nous pouvons donc coder les obliques dans notre structure.

2) LA VERIFICATION DE GARDE

Pour la vérification des gardes intérieures et extérieures, cette modification ne change pas le principe de base du traitement. Le parcours de la structure et la recherche de tangence de rectangles ne changent pas. Les différences résident dans l'existence éventuelle de la tangence pour un triangle, et dans le calcul des distances. Le traitement n'est pas modifié, il est simplement complété.

3) LES OPERATEURS LOGIQUES

Pour la réalisation des opérations logiques, le problème est complexe. En effet, pour les opérateurs rectangles tels que le XOR, on traite tous les cas d'intersections de deux rectangles de façon particulière. Ceci est possible, car les configurations différentes sont au nombre de 16. En ajoutant les triangles-rectangles, ce nombre est passé de 16 à plus d'un millier. Il est donc impossible de conserver ce mode de fonctionnement. Pour résoudre ce problème, la seule solution envisageable est la suivante :

- Pour les intersections de rectangles, on conserve ce principe.
- Pour les intersections mettant en oeuvre un ou deux triangles, on revient à une description des figures sous formes de segments. A partir de là, on calcule la ou les formes résultantes que l'on découpe à nouveau sous forme de rectangles et triangles-rectangles.

Cette solution ne met pas en cause le codage rectangle, car la majorité des figures constituant un masque sont des rectangles. On conserve donc les performances de la structure initiale. La détermination des intersections des segments n'est pas trop pénalisante dans la mesure où le nombre maximum de segments à traiter à un instant donné est de sept, ce qui limite fortement les possibilités.

Un autre point à considérer est la découpe des figures résultantes. A ce niveau, il est possible d'obtenir des figures non "découpables". Dans le cas d'une vérification de règles de garde, ceci n'est pas gênant. En effet, si un masque est en erreur, tous les niveaux résultants de combinaisons avec ce masque le seront aussi. Pour les autres cas, l'utilisateur a la possibilité de demander au programme une approximation de la forme avec la précision qu'il désire.

4) LES FONCTIONS PARTICULIERES

Pour toutes les fonctions particulières sauf une, le traitement est complété sans remettre en cause le principe de base du fonctionnement de ces opérateurs.

La seule fonction radicalement modifiée est la fonction de dilatation d'un masque. Dans le codage rectangle uniquement, la dilatation d'un rectangle se faisait de manière sélective en fonction de ses tangences avec ses voisins. Pour pouvoir garantir la validité du résultat, cette solution n'est plus applicable. Au détriment de la rapidité de traitement, le principe suivant est utilisé :

- On dilate les éléments sans s'occuper des tangences.
- On effectue un OU du masque sur lui même, ce qui a pour effet de supprimer les recouvrements.

Ce ralentissement de l'opérateur de dilatation n'est pas trop gênant car cet opérateur est peu employé. Dans le cas d'une technologie NMOS 6 μm, par exemple, cette fonction n'est utilisée qu'une seule fois.

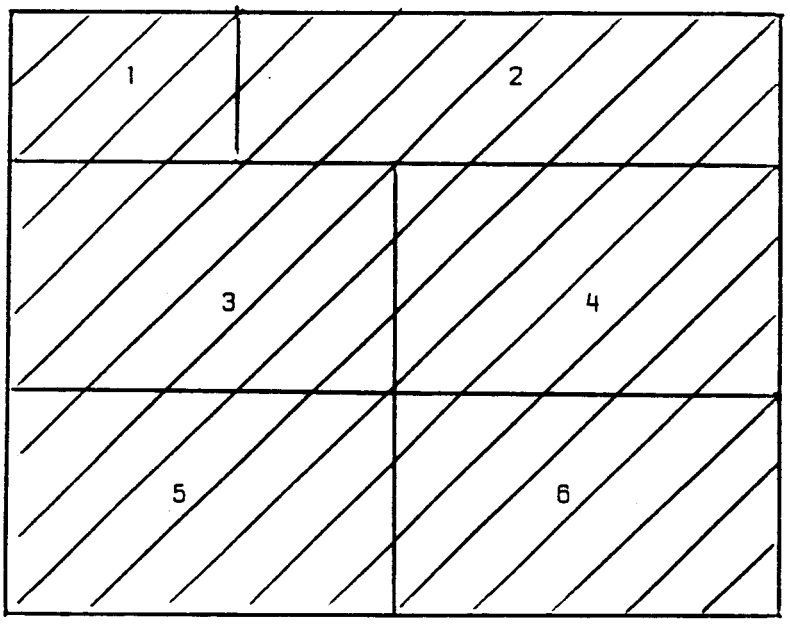
VII- LA HIERARCHISATION DU VERIFICATEUR

Dans ce que nous avons vu jusqu'à maintenant, la vérification des règles de garde se fait sur l'ensemble du circuit. Les temps de calcul sont donc très importants. Le but de la hiérarchisation du DRC est d'optimiser les temps de réponses.

1) PRINCIPE DE LA VERIFICATION HIERARCHIQUE

Partant de la constatation qu'un circuit est généralement réalisé à partir de figures qui se répètent sur la surface un grand nombre de fois, il est possible de n'utiliser que des figures de base qui, une fois vérifiées, seront considérées comme correctes. Leur implantation dans l'ensemble du dessin n'utilisera plus que les contours de ces figures de base nécessaires à la vérification totale du circuit.

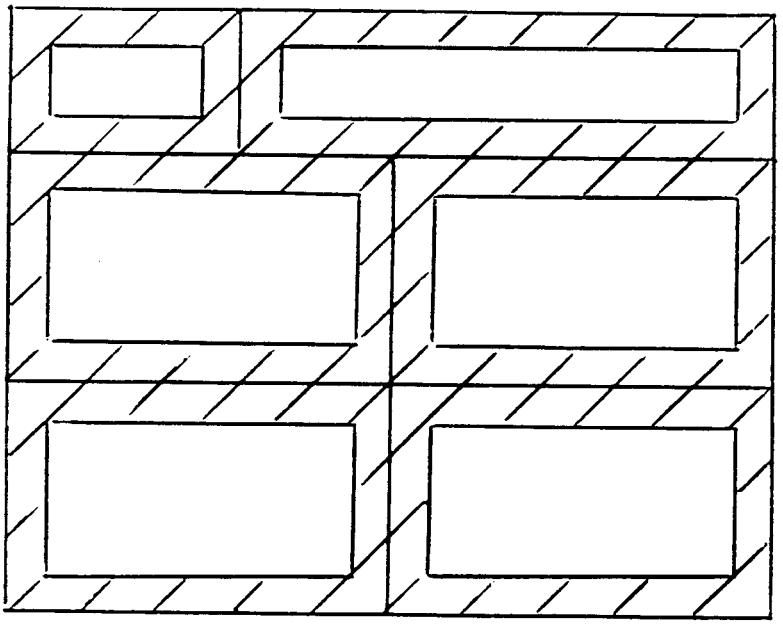
Figure 51 :



3, 4, 5 et 6 sont identiques

On vérifie intégralement 1, 2 et 3

Figure 52 :



Les vides sont considérés comme vérifiés, on n'utilise donc que l'assemblage des contours des figures pour la vérification de niveau supérieur, ce qui amène un gain de temps appréciable. De plus, cette découpe fait travailler le vérificateur sur des ensembles plus réduits du circuit, ce qui permet un gain de place mémoire et d'espace disque.

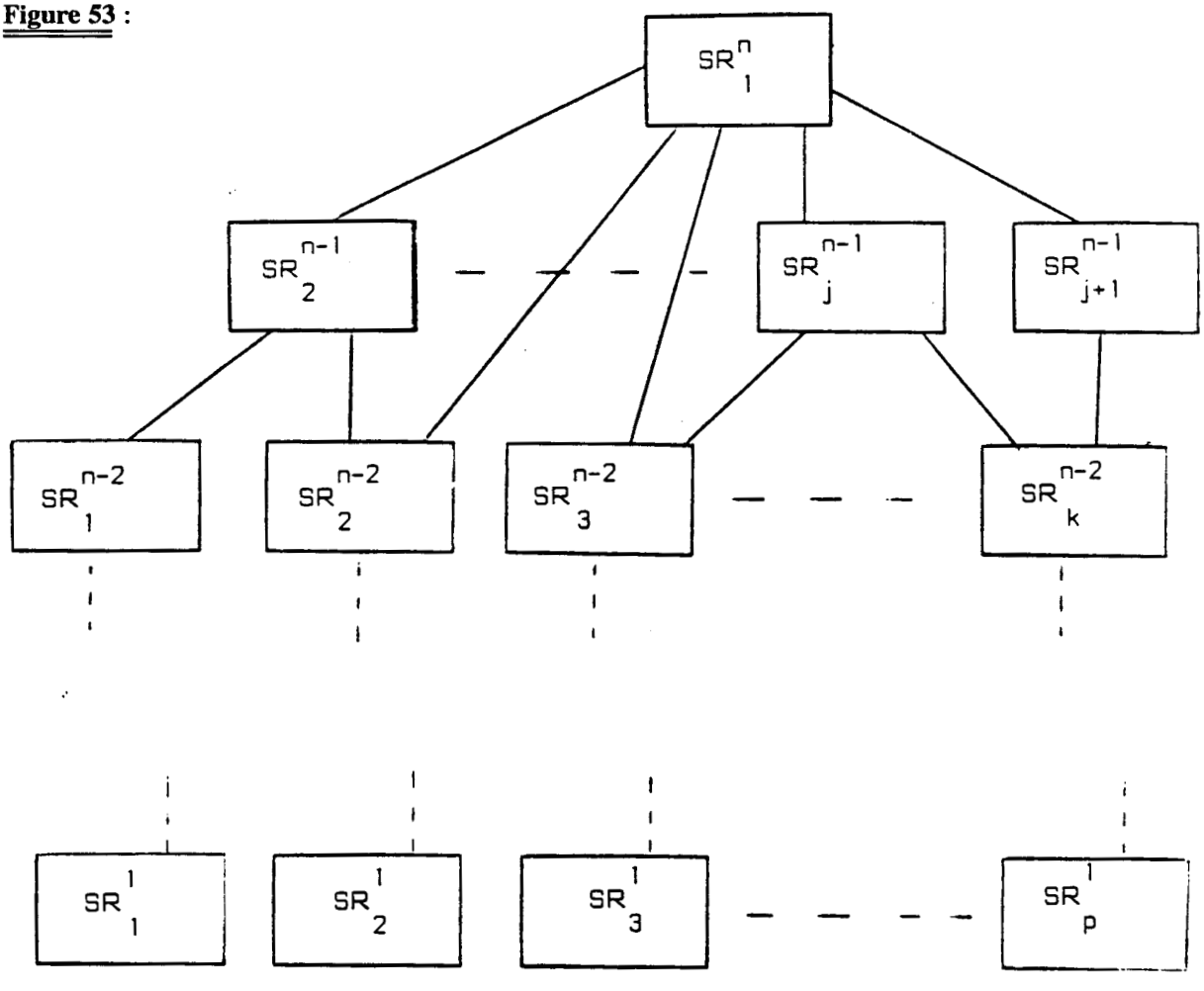
Le gain de temps n'est effectif que si les cellules ont une surface suffisamment importante. Les tests du programme montrent que la surface totale de la cellule doit être au minimum trois fois supérieure à la surface du bord restant après découpe. Si cette condition n'est pas respectée, le temps de découpe est supérieur au temps gagné pendant la vérification.

Prenons l'exemple du format GDS2.

Le fichier initial est composé des structures (blocs) référencées (AREF : matrice de structures ou SREF : structure référencée) qui appellent d'autres structures de niveau inférieur contenues dans le fichier. On a donc une arborescence du type suivant :

S.R = structure référencée (AREF ou SREF)

Figure 53 :



Une structure de niveau i peut appeler une structure de niveau inférieur. Les structures de niveau 1 sont nécessairement des "boundaries" (polygones fermés). Sans hiérarchisation du vérificateur, pour vérifier un tel fichier, on génère l'ensemble des points le composant avant de le soumettre au DRC. C'est une possibilité que l'on conserve. Dans le cas de la découpe des éléments, on doit d'une part vérifier que la figure générée est bonne, et réaliser une découpe éventuelle de celle-ci. On soumet les unes après les autres les structures référencées non encore découpées au vérificateur en commençant par le niveau le plus bas (niveau 2 car le niveau 1 est toujours bon). On génère au fur et à mesure, s'il y a lieu, les figures trouées nécessaires à la construction d'une structure de niveau supérieure. Comme une structure peut être référencée plusieurs fois, on s'attache à ne réaliser qu'une et une seule découpe et donc une et une seule vérification.

2) LA DECOUPE DES CELLULES

Une structure est composée de contours fermés et orientés. Lors de la découpe, on réalise donc l'intersection d'un contour et d'un autre ensemble de points.

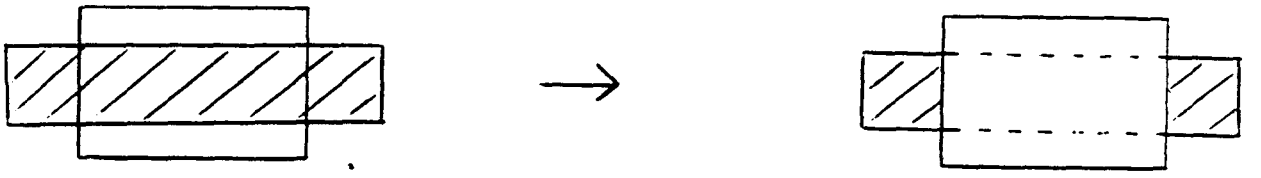
Trois cas de figure se présentent :

- On obtient un contour fermé et orienté

Figure 54 :

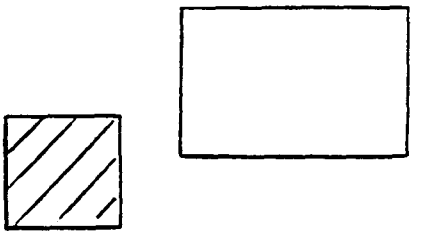
- On obtient plusieurs contours

Figure 55 :



- La figure résultante est le contour initial

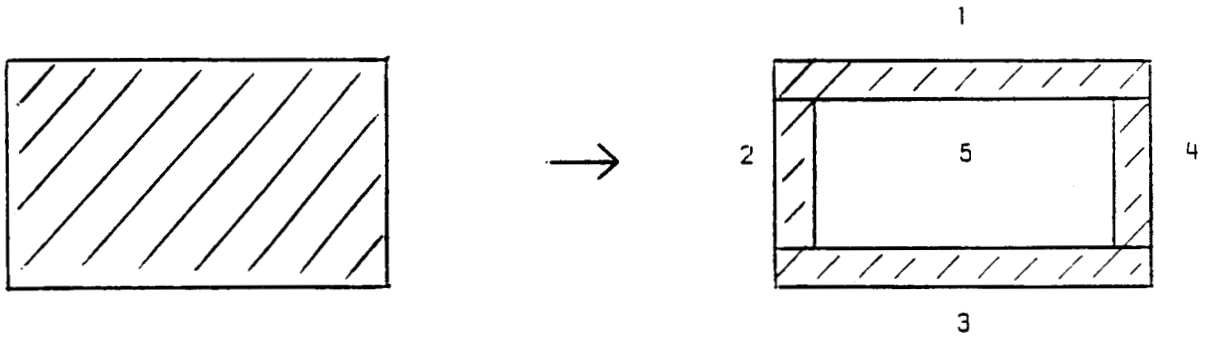
Figure 56 :



Dans ces 3 cas, on doit fermer le contour avec la même orientation que précédemment pour ne pas modifier le format d'écriture.

Une des contraintes imposées par le vérificateur précédemment décrit, est qu'on ne peut pas lui faire traiter directement une structure perforée, car il considère le circuit comme formé d'une partie centrale rectangulaire complétée d'un bord. Il faut donc lui fournir 4 éléments afin qu'il puisse faire une analyse sans générer d'erreurs dues à la découpe.

Figure 57 :

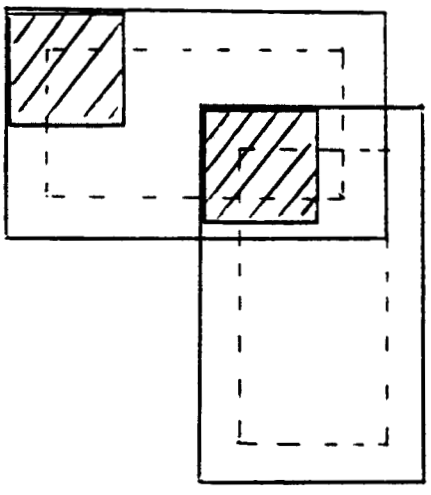


Nous devons donc générer, après création du vide à la suite de la vérification du bloc, le type suivant de SREF ou AREF, composé de 4 éléments.

Pour réaliser cette découpe, on détermine d'abord l'encombrement rectangulaire de la figure, puis on laisse une marge suffisante pour la vérification (garde maximale) et on retire le centre.

Cette découpe implique qu'on ne peut pas permettre par la suite, une violation de la zone 5 lors de la construction d'une structure référencée de niveau supérieur à partir de cette structure vérifiée et simplifiée.

Figure 58 :

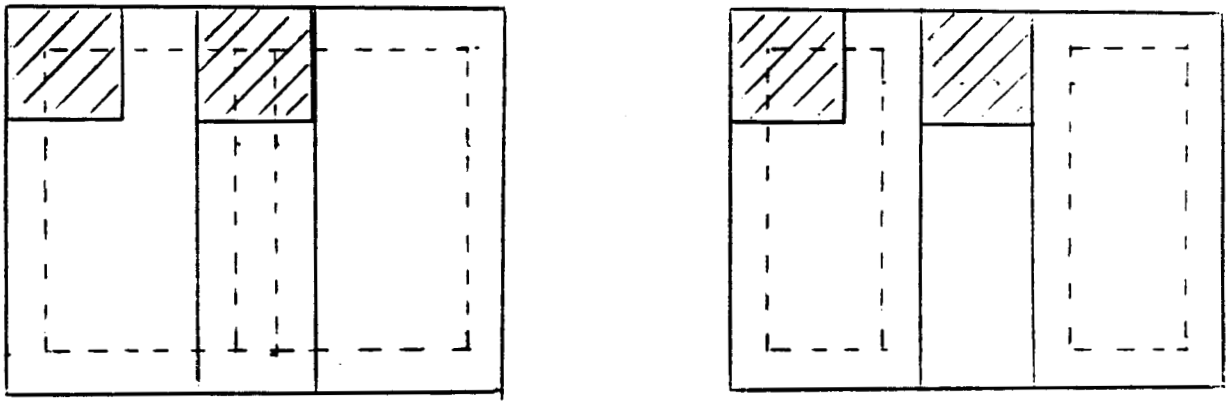


Donc si on a une interpénétration de structure, par exemple suite à une rotation ou une translation (exemple précédent), on doit garantir la validité de la vérification. On peut remarquer, d'après la construction du fichier initial, qu'une structure de niveau supérieur peut appeler une structure d'un quelconque niveau inférieur (ex : un polygone). Il en résulte que l'apparition du recouvrement se fera en fin de construction de la figure. Il faut donc réaliser une analyse de la figure avant la mise en oeuvre des fonctions de découpe. Dans la cas où il y a superposition, 2 solutions sont envisageables :

- Changement du "bord"

On recalcule un nouveau "bord intérieur" de découpe pour que l'assemblage des deux bords contienne la totalité du recouvrement.

Figure 59 :

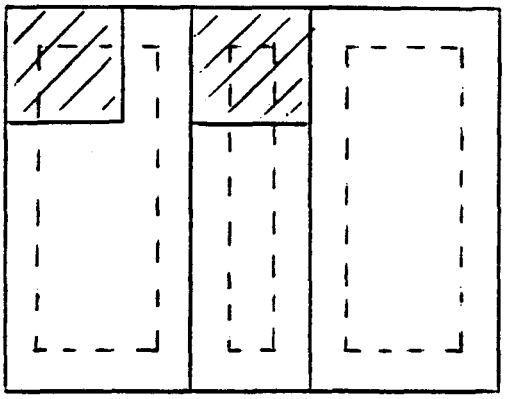


- Création d'une figure

On considère la superposition comme une nouvelle figure qu'il faut vérifier au même titre qu'une autre.

L'exemple précédent nous donnerait :

Figure 60 :



Cette solution est plus complexe que la précédente puisqu'elle fait apparaître une nouvelle figure. D'autre part, on remarque que les deux solutions imposent de connaître parfaitement la structure appelée (point par point). On doit donc redescendre l'arborescence jusqu'au niveau où la superposition a eu lieu, puis la reconstruire avec une nouvelle découpe. Il paraît plus judicieux d'analyser d'abord le circuit en terme d'encombrement maximum, puis d'en déduire les découpes maximales à réaliser dans chaque structure référencée.

VIII- COMPARAISON AVEC LES PRODUITS DU MARCHE

Une présentation de deux des produits les plus répandus est donnée en annexe 5. Il s'agit de DRACULA de ECAD et de UDRC de NCA. Nous ne ferons ici qu'une comparaison des vérificateurs de règles de garde. DRACULA représente les masques comme un ensemble de trapèzes, UDRC comme un ensemble de segments. Nous comparons ces logiciels sur les points suivants :

- Complexité des vérifications effectuées.

Sur ce point, le logiciel le plus complet est UDRC. Il est le seul qui autorise des vérifications asymétriques dépendant de l'orientation des lignes de connexions. Ces règles deviennent de plus en plus fréquentes au fur et à mesure que diminuent les dimensions des éléments. Sur le plan des possibilités de vérification, notre produit est équivalent à DRACULA.

- Facilité d'encodage.

Pour utiliser ces produits, le concepteur doit encoder sa technologie, c'est à dire qu'il doit, pour chaque règle, décrire l'ensemble des opérations à effectuer et les niveaux mis en oeuvre. Sur ce point, notre produit est le plus simple. A titre indicatif, signalons qu'une personne ne connaissant pas le logiciel a mis une demi-heure pour réaliser l'encodage d'une technologie CMOS 2 μm.

UDRC présente l'inconvénient d'être très complexe pour le codage. Ceci est lié aux énormes possibilités de ce produit. Ce point n'est cependant pas trop gênant car un encodage de technologie est une tâche peu fréquente.

- Temps de traitement.

Sur ce point, le logiciel le plus rapide est DRACULA, suivi de près par le notre (moins de 10% de temps de traitement supplémentaire) et loin devant UDRC (deux fois plus lent que DRACULA). Les tests ont été réalisés sur un circuit en technologie CMOS 2 μm comportant environ 10 000 transistors.

- Occupation mémoire.

Sur ce point, seul notre produit peut s'exécuter sur une machine ayant moins de quatre méga-octets de mémoire centrale. Le plus gourmand des produits est UDRC.

IX- CONCLUSION

Notre produit présente donc un certain nombre d'avantages. Parmi ceux-ci citons :

- L'occupation mémoire qui est inférieure à 1 M octets. La méthode de découpe des masques de circuits intégrés sous forme de rectangles permet donc de réduire considérablement l'espace mémoire centrale requis. L'implantation de ce logiciel sur du matériel de type micro-ordinateur est tout à fait réaliste.

- L'évolution du temps de calcul en fonction du nombre de rectangles du dessin qui est linéaire. Si on prend une évolution en n, le facteur multiplicatif est inférieur à 1,09.

A titre indicatif, la vérification complète d'un filtre à capacités commutées comportant environ 1000 transistors et 300 capacités unitaires (soit 15 000 rectangles) a consommé 15 minutes CPU sur un IBM 4341 modèle 2.

Des tests ont aussi été effectués pour une technologie CMOS 2 μm. Pour un circuit comportant environ 30 000 rectangles et pour une vérification composée de 60 opérations élémentaires, le temps CPU utilisé était de l'ordre de 30 minutes.

- La facilité d'encodage d'une nouvelle technologie

- La grande variété des vérifications possibles qui est comparable à celle offerte par des produits mondialement utilisés dans l'industrie.

L'EXTRACTEUR DE SCHEMAS

Les concepts mis en oeuvre pour la réalisation d'un extracteur de schémas (c'est à dire le passage du dessin des masques au schéma électrique) sont présentés.

L'analyse des différents types de vérifications souhaitables fait apparaître trois fonctions différentes. Les simplifications apportées aux modèles de bases afin de rendre la réalisation possible sont expliquées en détail.

Les opérateurs mis en oeuvre dans cet outil ainsi que leur couplage avec le vérificateur de règles de gardes sont présentés.

Le plan du chapitre est le suivant :

- L'extraction de schéma pour comparaison, présentant les besoins d'une extraction pour effectuer une comparaison de schéma.

- L'extraction fine pour simulation, qui en justifie la nécessité pour une simulation des chemins critiques.

- La comparaison de schémas, présentant les problèmes liés à la comparaison de deux schémas électriques.

I- INTRODUCTION

Lors de la réalisation du dessin des masques d'un circuit, de nombreuses erreurs peuvent se glisser. Ces erreurs peuvent être de deux types :

- géométriques. Le dessin ne respecte pas les règles de garde. Ces erreurs sont détectées par le vérificateur de gardes.

- fonctionnelles. Le dessin est géométriquement correct, mais le circuit ainsi implanté est différent de celui désiré (oubli d'un contact, d'une connexion...). Ces erreurs peuvent engendrer des disfonctionnements importants du circuit, et doivent donc être corrigées avant la fabrication. La vérification manuelle est impossible, il faut donc recourir à une extraction de schéma permettant, par la suite, une simulation ou une comparaison du circuit afin de valider le travail effectué.

L'objectif était, à partir des masques d'un circuit intégré, de réaliser un programme permettant de reconstituer le schéma électrique. Par comparaison avec le schéma électrique désiré, il devient possible de détecter les éventuels oublis ou erreurs de conceptions apparus lors du passage du schéma électrique au dessin des masques.

En entrée de ce programme, on fournit l'ensemble des rectangles et triangles-rectangles constituant chaque masque. Les fichiers d'entrée de l'extracteur sont les mêmes que ceux du vérificateur de gardes. Le nombre de masques varie avec la technologie employée.

En sortie, le programme fourni des fichiers analogues aux fichiers d'entrée du simulateur SPICE: liste des éléments (dipôles, quadripôles) et pour chacun, les numéros d'équipotentielles auxquelles sont connectées ses bornes, ainsi que ses caractéristiques électriques déduites de sa géométrie.

II- L'EXTRACTION DE SCHEMA POUR COMPARAISON

Quand on effectue une extraction de schéma dans le but de faire une comparaison, cette extraction est sommaire. En effet, on ne s'intéresse qu'aux éléments constituant le circuit, sans prendre en compte les parasites induits par la fabrication. Il suffit donc de retrouver les transistors, résistances, capacités et diodes implantés par le concepteur et de les interconnecter. Ce type de traitement nécessite les quatres étapes décrites dans les paragraphes suivants.

1) DETERMINATION DES ELEMENTS ELECTRIQUES

Les schémas suivants résument, pour deux technologies (CMOS et NMOS), la morphologie, les caractéristiques électriques qui peuvent être déduites de la géométrie, et les connexions des principaux quadripôles utilisés en micro-électronique.

Figure 61 : Technologie NMOS

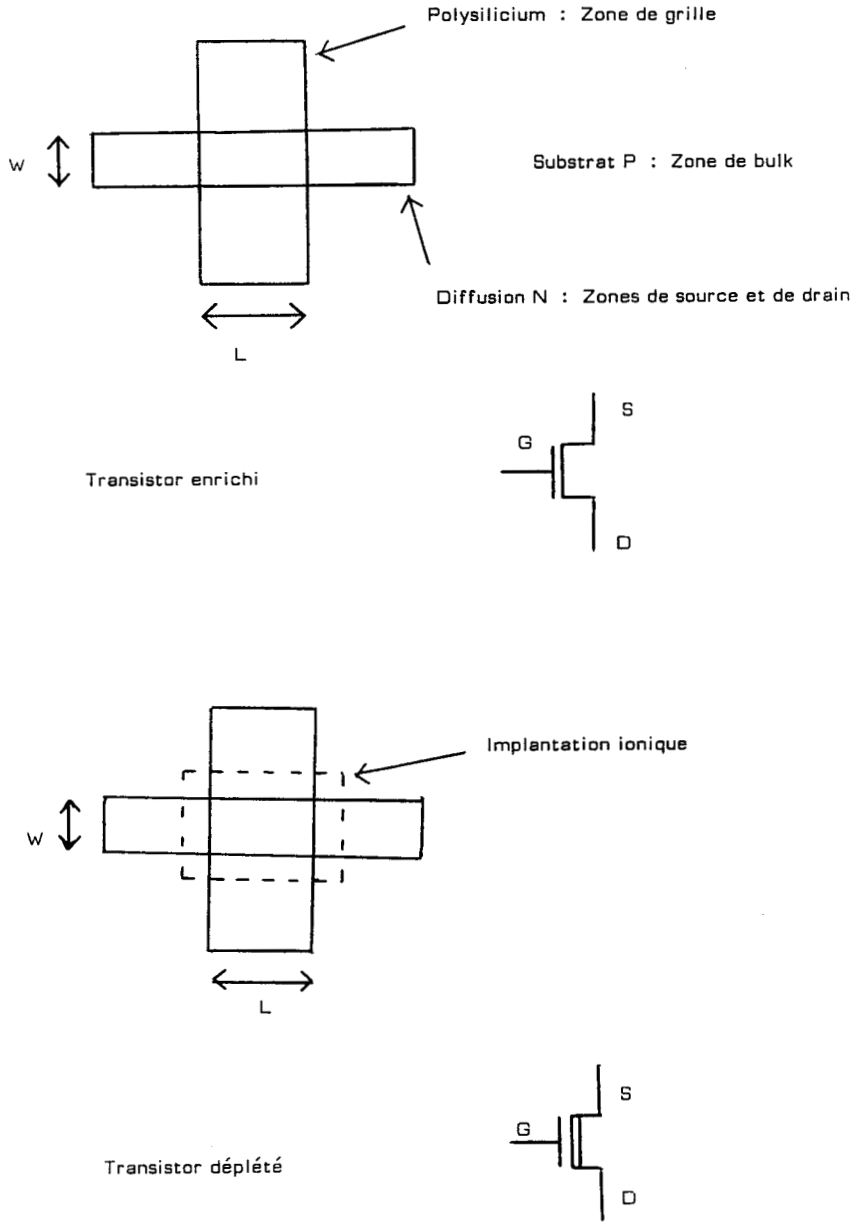
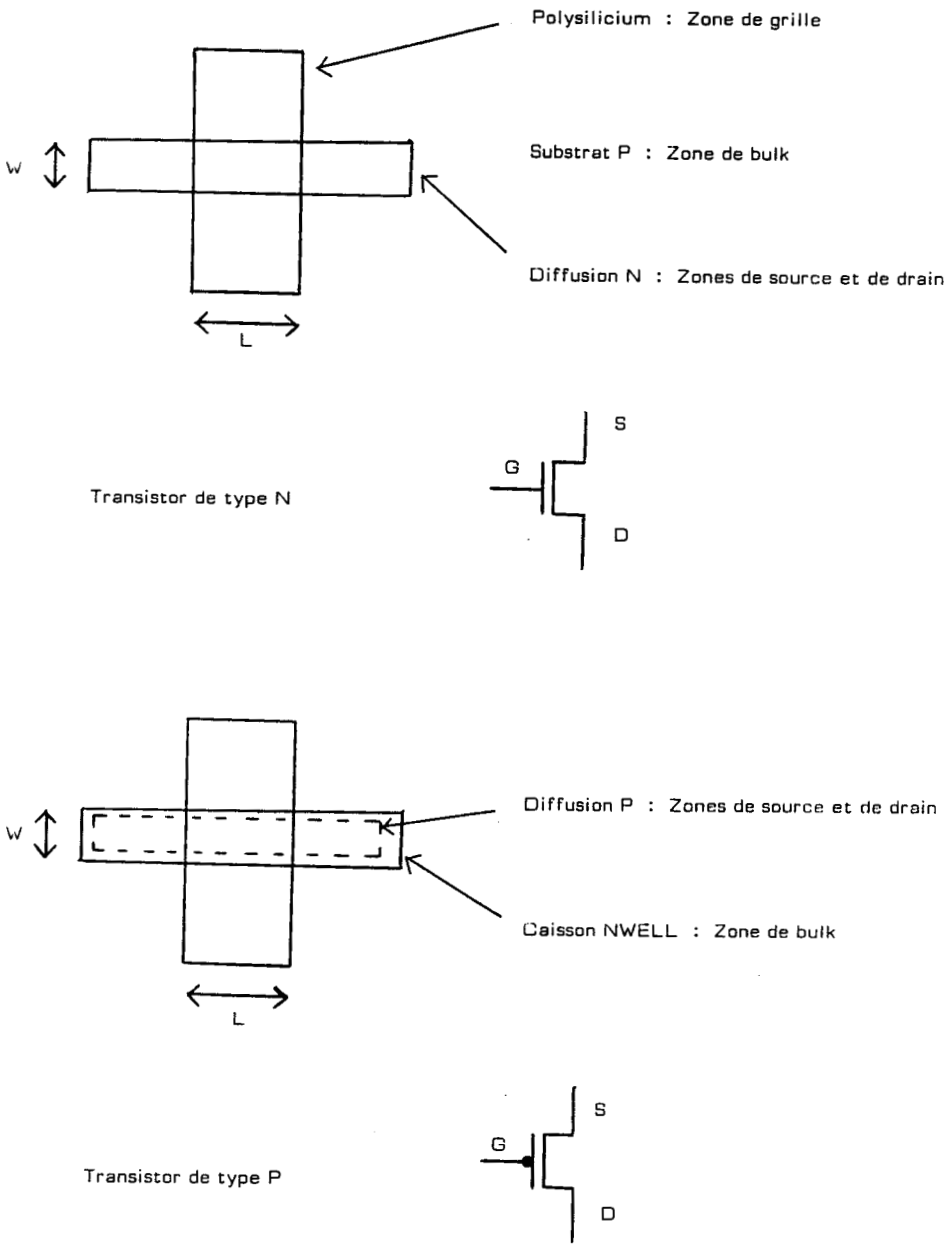


Figure 62 : Technologie CMOS



Dans les deux technologies, les résistances sont constituées de lignes de polysilicium ou de diffusion. Elles sont très rares et peuvent être ignorées pour la comparaison. Si ce point devient gênant, il suffit de générer un masque particulier pour ces éléments. Il en est de même pour les capacités qui sont constituées par superposition de deux niveaux.

Il n'y a que des éléments géométriques en entrée du programme. Tout élément extrait doit donc l'être géométriquement. Ainsi, en technologie NMOS, si on effectue un ET (intersection) entre les masques de polysilicium et de diffusion, on obtient un masque résultat contenant tous les transistors. Pour avoir les transistors déplétés, il faut encore effectuer un ET avec le masque des transistors et celui de l'implantation ionique. Pour obtenir les transistors enrichis, il faut effectuer la soustraction (MOINS) entre le masque des transistors et celui des transistors déplétés. On peut de même extraire les transistors P et N dans une technologie CMOS. Les fonctions logiques (ET, MOINS, NON, OU et XOR) utilisées pour la détermination des éléments électriques sont déjà disponibles dans le vérificateur des règles de garde.

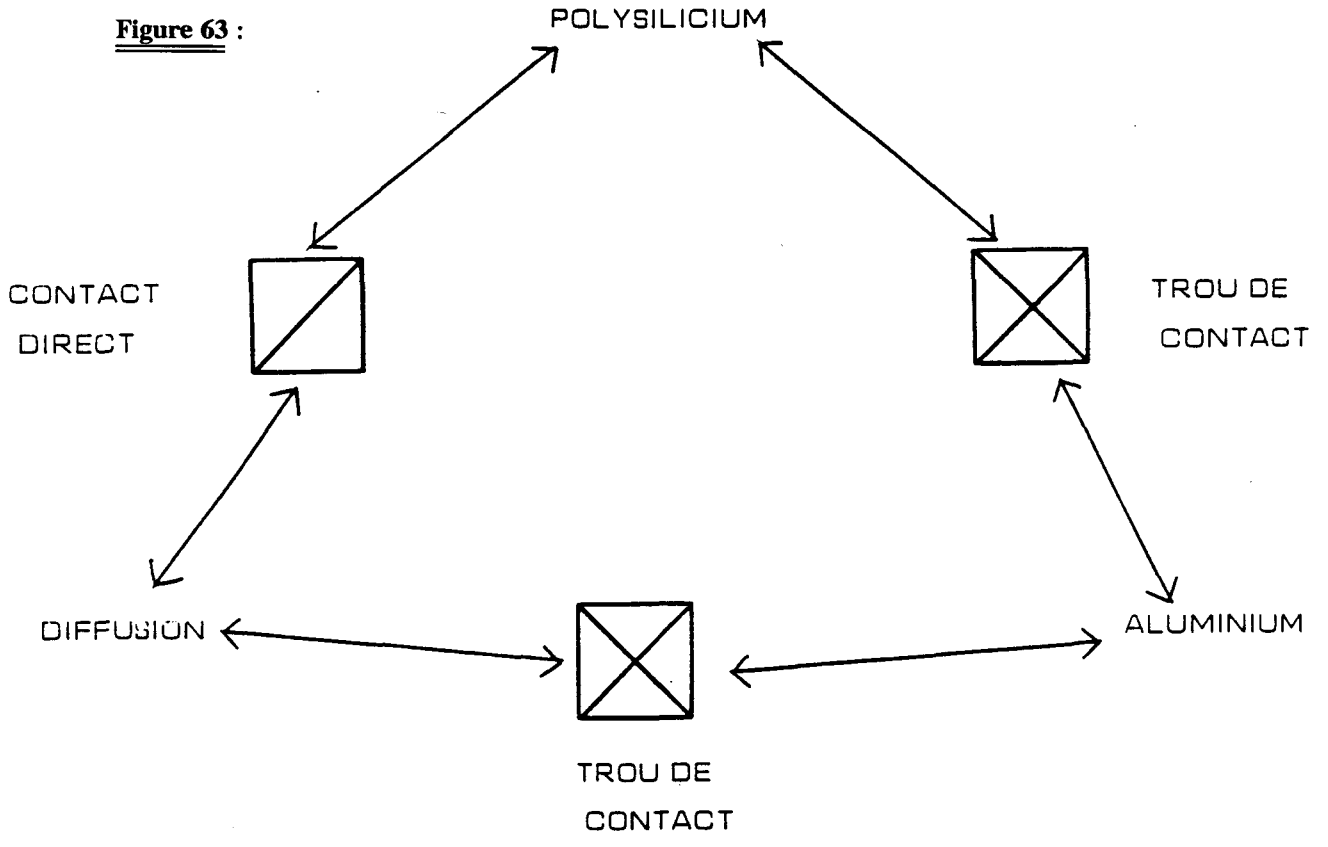
Dans le cas des dipôles, ou le concepteur fournit un masque spécial contenant ces éléments (lignes de polysilicium, diffusion...) ou un masque permettant de retrouver ces éléments peu nombreux, ou il faut implémenter une fonction spéciale qui les détectera (ligne de polysilicium ayant une résistance supérieure à une valeur donnée...). Nous avons retenu la première solution.

2) DETERMINATION DE LA CONNECTIQUE

Il existe au moins trois niveaux permettant de réaliser des connexions : le polysilicium, la diffusion et le métal. Dans certaines technologies, on peut trouver un deuxième niveau de métal ou de polysilicium.

Les interconnexions entre ces différents niveaux se font selon la figure 63.

Figure 63 :



Une fois que tous les éléments électriques ont été enlevés des masques origines, il ne reste que la connectique. Le problème est de repérer ces interconnexions afin de leur faire correspondre les éléments électriques.

En fait, il y a quatre problèmes à résoudre :

- L'extraction géométrique des éléments électriques
- Le repérage de la connectique
- L'attribution de la connectique aux éléments électriques
- La détermination des caractéristiques électriques de ces éléments

3) DETERMINATION DES EQUIPOTENTIELLES

La notion d'équipotentielle est indispensable lorsqu'il s'agit de décrire la schématique. En conséquence, il faut une fonction permettant de déterminer ces équipotentielles. Cette fonction numérote et chaîne les éléments constituant les équipotentielles d'un masque donné. Deux entités (rectangle ou triangle) d'un même niveau ayant une intersection non vide appartiennent à la même équipotentielle. Cette fonction permet de repérer la connectique sur un niveau. Nous avons vu que ces différents niveaux pouvaient être interconnectés (trou de contact, contact enterré...) mais que cet opérateur à lui seul ne résoud pas tous les problèmes d'identification de la connectique.

Il faut donc créer une fonction qu'on appelle "correspondance d'équipotentielles" qui permet de relier les équipotentielles de deux ou plusieurs masques. Ainsi, le problème d'identification de la connectique sera entièrement résolu. Chaque élément d'un masque (qui n'est pas un élément électrique) aura un numéro d'équipotentielle de masque et un numéro d'équipotentielle absolue dans l'ensemble des masques.

4) LA GENERATION DES RESULTATS

Nous avons vu que chaque type d'élément électrique correspond à un masque (masque origine ou masque déterminé par les fonctions logiques du DRC). Si on applique la fonction équipotentielle sur ces niveaux, chaque élément électrique reçoit un numéro d'équipotentielle puisque ces éléments n'ont pas d'intersection. Ce numéro peut servir d'identifiant.

Le problème suivant est le repérage des bornes des éléments. Pour cela, deux fonctions sont nécessaires :

- Une fonction "tangence" qui recherche, pour chaque élément électrique, deux équipotentielles tangentes. Cet opérateur permet, par exemple, de déterminer les sources et les drains des transistors ou les deux bornes d'une résistance.

- Une fonction "équivalence" qui recherche, pour chaque élément électrique, le numéro d'équipotentielle équivalente dans un autre masque. Cet opérateur permet, par exemple, de déterminer la grille d'un transistor ou les bornes d'une capacité.

Le dernier travail à faire avant l'édition des résultats est la détermination des caractéristiques électriques des éléments. Celles-ci ne peuvent être déduites que des caractéristiques géométriques des masques. Il faut donc créer une fonction par valeur à déterminer. Les principaux opérateurs effectuent le calcul des valeurs W (largeur du canal) et L (longueur du canal) des transistors, ainsi que le calcul des valeurs des résistances et des capacités.

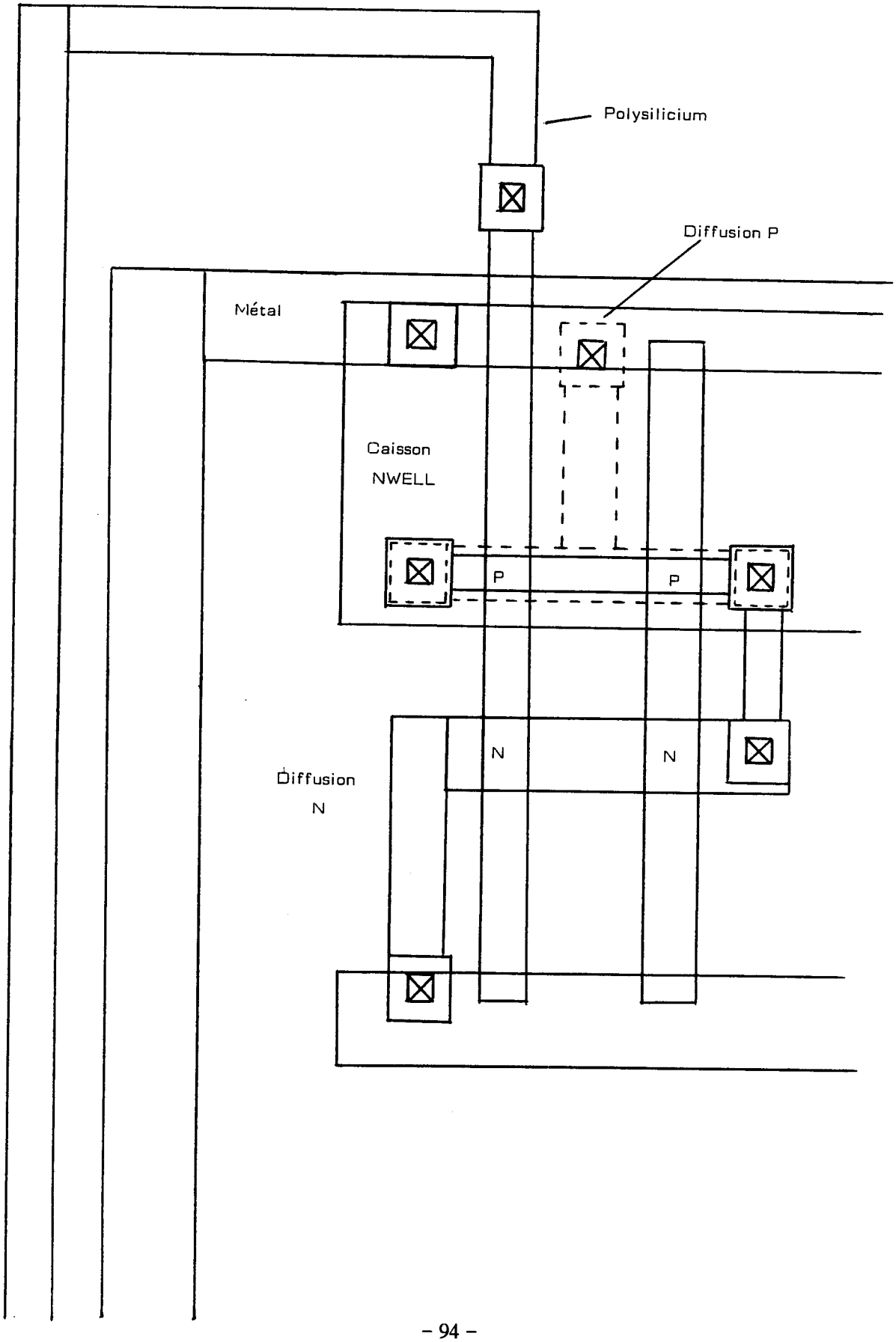
La fonction d'édition se contente de balayer ces différents résultats et de les écrire en clair sous le format SPICE.

5) EXEMPLE D'EXTRACTION

Voici un exemple de fichier de commande et la description des actions réalisées.

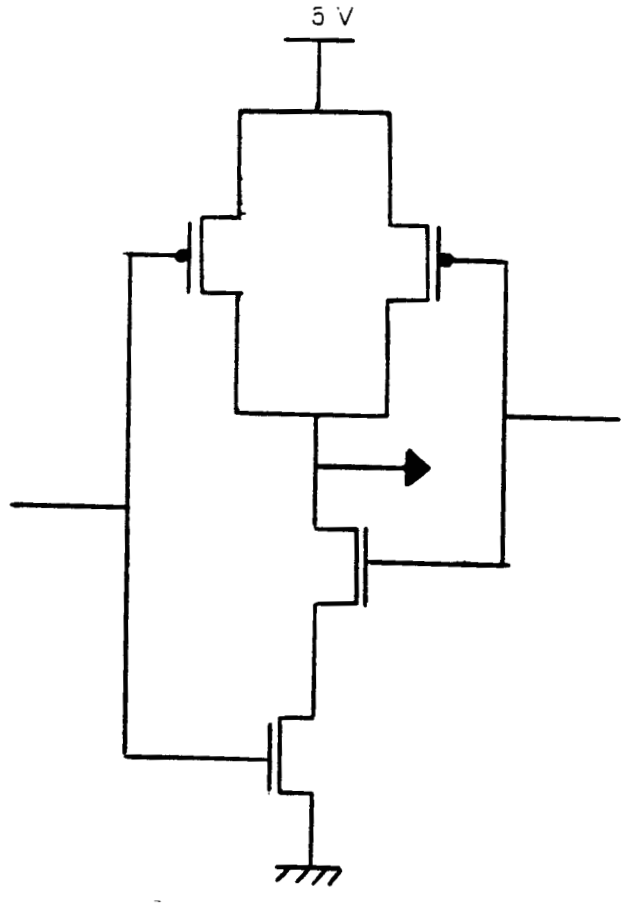
Le dessin des masques en entrée est donné à la figure 64.

Figure 64 :


Polysilicium
Diffusion P
Métal
Caisson
NWELL
P
P
N
N
Diffusion
N

Ce schéma représente une porte NAND en technologie CMOS, correspondant au schéma électrique, figure 65 :

Figure 65 :



Nous ne considérons que les transistors.
La première étape est la détermination des éléments électriques.

- Les transistors de type P

a = polysilicium ET diffusion p

aeqp = EQP a	repérage des transistors
pmos = FELMT aeqp 4 2	génération d'un fichier élément à quatre bornes et deux caractéristiques

- Les transistors de type N

b = polysilicium ET diffusion n
beqp = EQP b
nmos = FELMT beqp 4 2

- Suppression des éléments électriques pour obtenir la connectique

dp = diffusion p MOINS a — Zones sources et drains des transistors p

dn = diffusion n MOINS b — Zones sources et drains des transistors n

- Identification de la connectique

dpeqp = EQP dp
dneqp = EQP dn
polyeqp = EQP polysilicium
alueqp = EQP aluminium
trceqp = EQP trou de contact
bk1p = EQP nwell
bk1n = EQP (NON nwell)
bk2p = EQP (nwell ET diffusion n)
bk2n = EQP ((NON nwell) ET diffusion p)

- Mise en correspondance des différentes équipotentielles

tc = CEQP dneqp trceqp
tc = CEQP dpeqp trceqp
tc = CEQP polyeqp trceqp
tc = CEQP bk1p bk2p
tc = CEQP bk1n bk2n
tc = CEQP bk2p trceqp
tc = CEQP bk2n trceqp
tc = CEQP alueqp trceqp

- Attribution des numéros d'équipotentielles aux bornes des éléments

- grilles des transistors P et N

polyeqp = EQV pmos 1 — Position 1 du fichier transistors

polyeqp = EQV nmos 1

- sources et drains

dpeqp = TGT pmos 2 3 — Positions 2 et 3 du fichier transistors

dneqp = TGT nmos 2 3

- bulk

bk1p = EQV pmos 4 — Position 4 du fichier transistors

bk1n = EQV nmos 4

- Edition des résultats

EDIT nmos tc

EDIT pmos tc

III- L'EXTRACTION FINE POUR SIMULATION

Quand on désire simuler de manière précise le comportement d'une partie sensible d'un circuit, il convient de prendre en compte les éléments parasites induits par la fabrication. Cette extraction fine nécessite des traitements plus complexes et plus longs que l'extraction pour comparaison. Les différentes étapes sont expliquées dans les paragraphes suivants.

1) LES ELEMENTS PARASITES

Il existe trois grandes catégories d'éléments parasites :

a- Les résistances

Les résistances parasites sont générées par les lignes de connexions. Ainsi, en technologie NMOS 6 μm, une ligne de polysilicium présente une résistance de 20 ohms par carreau (on appelle carreau un carré d'un niveau quelconque. Les résistances d'un carré de 1 μ de coté et celle d'un carré de 200 μ de coté sont les mêmes et valent toutes deux la valeur de la résistance par carreau soit 20 ohms dans le cas du polysilicium). Cette résistance est de 22 ohms pour la diffusion et de 0,03 ohms pour l'aluminium. On s'aperçoit rapidement que ces résistances ne sont pas négligeables pour de longues lignes de polysilicium ou de diffusion. Ainsi, une connexion en polysilicium de 6 μm de large et de 30 μm de long représente 30:6 soit 5 carreaux, ce qui nous donne une résistance de 110 ohms.

b- Les capacités

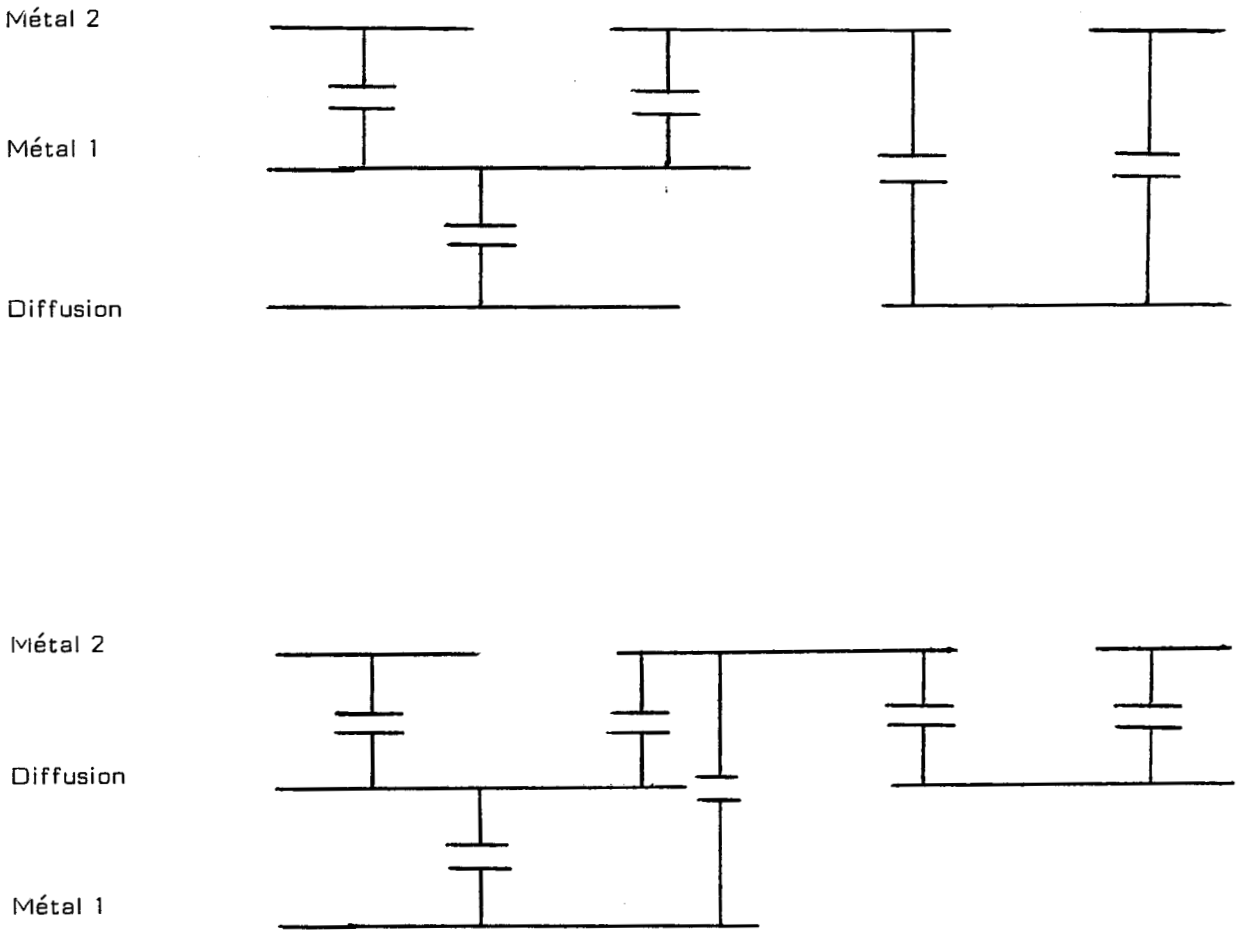
Les capacités parasites sont générées par superposition des différentes couches technologiques. Par exemple, en technologie NMOS 6 μm, la capacité du polysilicium par rapport au substrat est de 0,5 10^{-4} pF/μm². La capacité générée par la superposition d'une ligne d'aluminium et d'une ligne de diffusion est de 0,6 10^{-4} pF/μm². Elle a la même valeur pour une superposition d'aluminium et de polysilicium. Ces capacités sont très importantes pour la détermination de la fréquence maximale de fonctionnement du circuit. En effet, par combinaison des résistances et capacités parasites, on obtient des réseaux RC parasites, qui limitent fortement la vitesse du circuit.

c- Les diodes

Les diodes parasites sont générées par les lignes de diffusion. On trouve des diodes en inverse entre le substrat et les connexions en diffusion. Ces diodes ont une capacité de jonction de 1,1 10^{-4} pF/μm² pour une technologie NMOS 6 μm. Ce phénomène induit donc des couplages capacitifs entre, par exemple, les différentes bornes d'un transistor. Ces parasites contribuent aussi à limiter la fréquence de fonctionnement du circuit. On remarque que la définition des diodes est très proche de celle des capacités. Nous traitons donc les diodes comme des capacités, sauf lors de l'édition des résultats où les capacités entre la diffusion et le substrat sont transformées en diodes.

Ces éléments parasites dépendent de l'ordre de fabrication des différents niveaux. En effet, en fonction de l'ordre de superposition des masques, les éléments parasites générés peuvent être très différents, comme le montre l'exemple suivant.

Figure 66 :

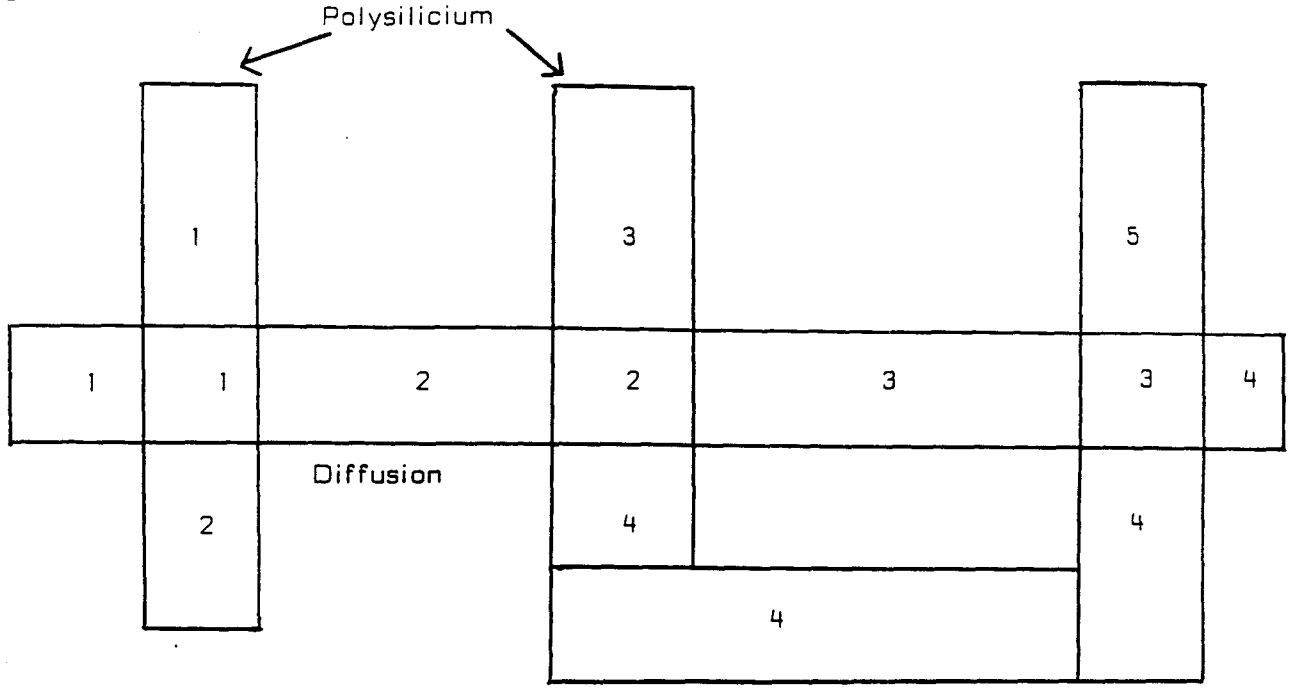


Ce schéma explique pourquoi il est nécessaire de connaître l'ordre de fabrication des masques, même si la deuxième configuration est une aberration technologique.

2) LA DETERMINATION DES NOEUDS INTERMEDIAIRES

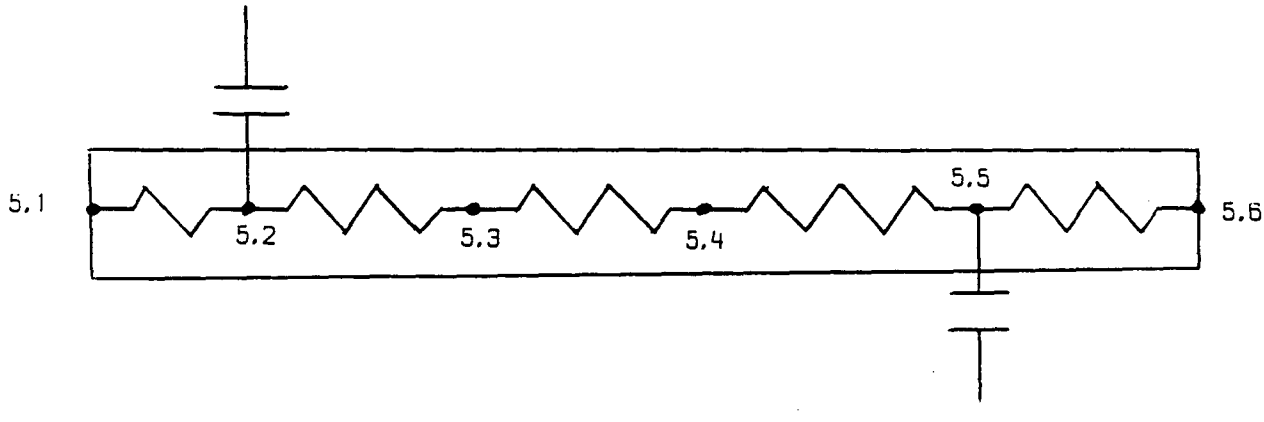
Pour réaliser une extraction fine, l'utilisateur doit d'abord identifier les composants du circuit. Cette identification se fait par l'intermédiaire des fonctions logiques, comme pour une extraction en vue d'une comparaison. De la même façon, on enlève ces éléments afin de déterminer la connectique du circuit. Une fois ceci réalisé, nous appliquons la fonction équipotentielle sur les masques résultants (composants et connectiques). Cet opérateur nous fournit des équipotentielles dites primaires, puisqu'elles ne prennent pas en compte les éléments parasites.

Figure 67 :



La détermination des équipotentielles inter-niveaux dues aux trous de contact s'effectue en appliquant ensuite la fonction correspondance d'équipotentielles sur les différents niveaux. Ceci permet d'obtenir les équipotentielles primaires dites absolues, à partir desquelles nous allons travailler pour ajouter les éléments parasites. Pour cela, chaque équipotentielle absolue est divisée en lui associant des équipotentielles secondaires qui seront, en fait, les points de jonction des réseaux R (résistances), C (capacités) et D (diodes). Chaque noeud sera défini par un numéro d'équipotentielle primaire suivi d'un indice d'équipotentielle secondaire.

Figure 68 :



La détermination des équipotentielles secondaires se fait par calcul des intersections entre les différents niveaux. Par exemple, une ligne de polysilicium traversée par une ligne de métal devra être coupée en trois morceaux, un partie de polysilicium sur substrat, une partie de polysilicium entre métal et substrat et à nouveau une partie de polysilicium sur substrat. Ceci se complique lorsqu'on travaille avec une technologie avec deux niveaux de métal et deux niveaux de polysilicium.

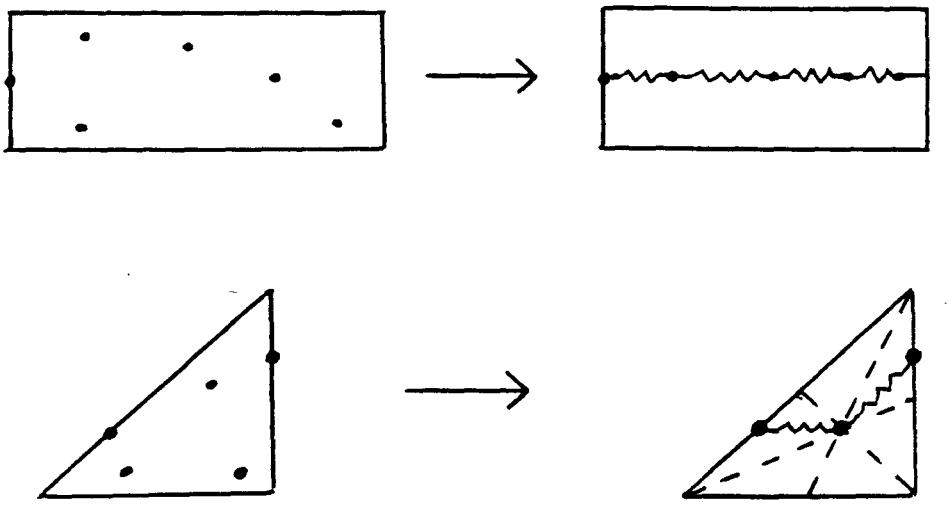
On obtient donc deux types de noeuds :

- Les noeuds de "tangence". C'est le cas pour deux rectangles tangents d'un même niveau.

- Les noeuds "capacitifs". Ces noeuds sont obtenus par superposition de couches technologiques différentes.

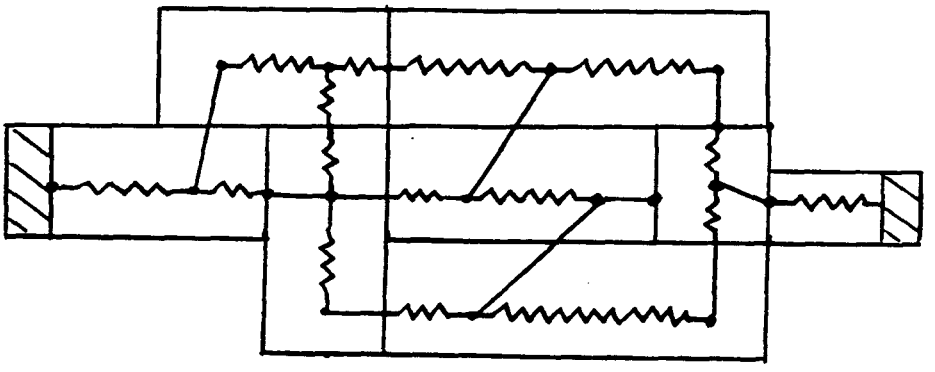
Ces noeuds peuvent se trouver n'importe où sur la surface de la forme (rectangle ou triangle) en cours de traitement. Pour résoudre ce problème, lorsque la forme est un rectangle, l'ensemble des noeuds est projeté sur la médiane. Les résistances sont insérées entre les différents noeuds projetés. Lorsque la forme est un triangle, les noeuds de "tangence" ne bougent pas, mais les noeuds "capacitifs" sont projetés sur le centre de gravité du triangle. Les résistances sont insérées entre le centre de gravité et les noeuds tangents.

Figure 69 :



Il est à noter que la détermination des tangences et des noeuds tangents n'est pas un problème simple. Voici un exemple montrant les limites de notre modèle.

Figure 70 :



Le modèle choisi permet de simuler corrrectement des lignes simples dans lesquelles une forme est tangente le plus souvent à deux autres formes. Par contre, tout amas de formes sera très mal simulé (enchevêtrement de noeuds connectés). Cette limitation peut être évitée par l'emploi de la fonction de regroupement de petites formes en forme de surface supérieure disponible dans le vérificateur de garde.

3) LA CONSTITUTION DES ELEMENTS

Une fois que toutes les équipotentielles secondaires sont déterminées, nous pouvons passer à l'étape de génération des éléments parasites.

Nous connaissons, pour chaque niveau, tous les noeuds intermédiaires, pour chaque forme le composant. Il suffit maintenant de connecter les différents niveaux entre eux. Pour la génération des capacités (ou diodes), l'utilisateur doit indiquer avec précision l'ensemble des masques sur lesquels on doit effectuer la recherche des capacités. Les masques doivent être introduits en débutant par le niveau le plus haut (hauteur par rapport au substrat) et en allant par niveaux décroissants.

Pour la génération des résistances, le traitement dépend du type de la forme en cours de traitement.

- La forme traitée est un rectangle. Dans ce cas, on insère simplement une résistance entre chaque noeud. Sa valeur est de L/W carreaux où L est la distance entre les noeuds et W la largeur de la ligne.

- La forme traitée est un triangle. Dans ce cas, un calcul plus précis doit être fait afin d'attribuer une valeur acceptable de résistance. En divisant le triangle en carreaux unités, on peut introduire un réseau de résistances. Les calculs effectués sur le schéma équivalent montrent, qu'en moyenne, la résistance entre deux points périphériques du triangle tend vers R (valeur de la résistance pour un carreau). On insère donc une résistance de R:2 entre tous points périphériques et le centre de gravité.

Lorsque ces éléments parasites sont générés, la suite du traitement (connexion des transistors...) est identique à l'extraction en vue d'une comparaison. Il en est de même pour la fabrication du fichier de sortie en format de type SPICE.

IV- LA COMPARAISON DE SCHEMAS

1) DESCRIPTION DU PROBLEME

Le but de la comparaison de schémas est de déterminer les éventuelles différences entre le circuit désiré et le circuit implanté sur silicium. Le comparateur dispose en entrée d'une netlist décrivant le circuit théorique, et d'une netlist décrivant le circuit réel. Le circuit théorique est fourni par le concepteur, le schéma réel est extrait à partir des masques.

Le problème est donc de comparer ces deux listes et de détecter les éventuelles erreurs. Les trois grandes catégories d'erreurs qu'on peut rencontrer sont :

- Le nombre de composants du schéma est différent. Il faut essayer de fournir à l'utilisateur une description la plus précise possible du ou des composants manquants ou trop nombreux.

- Les composants n'ont pas les mêmes caractéristiques. Par exemple, le nombre de transistors est le même pour les deux schémas, mais les valeurs W et L des transistors ne correspondent pas. Dans ce cas, il est très difficile de fournir une aide efficace au concepteur pour l'identification des transistors en erreur.

- Le nombre et les caractéristiques des composants sont les mêmes, mais les connexions sont différentes. C'est le cas d'erreur le plus fréquent. Le problème dans ce cas est le nombre d'erreurs possibles qui peut être très important. Il ne faut donc pas s'arrêter quand on a trouvé une erreur, mais il faut explorer toutes les solutions envisageables.

2) L'EXPLOSION NUMERIQUE

Le problème de la comparaison de schémas est le temps de calcul qui est très important. En reprenant les trois cas d'erreurs envisageables, les deux premiers ne posent pas de problèmes particuliers, et sont très rapides à détecter. La difficulté vient du troisième cas.

La source des ennuis est l'identification des équipotentielles. En effet, il est évident qu'un même transistor n'aura pas les mêmes numéros d'équipotentielles sur les deux schémas. Il faut donc faire la correspondance entre les noeuds des deux listes de connexions.

En prenant l'hypothèse d'un circuit comportant N transistors identiques, le nombre d'identifications différentes possibles se calcule de la manière suivante :

Chacun des quatre noeuds d'un transistor peut être connecté avec un des quatre noeuds d'un autre transistor. Ceci donne donc factorielle (4*N) possibilités. Pour 10 transistors, nous obtenons donc environ $8\ 10^{47}$ combinaisons possibles. On se rend compte aisément du temps de calcul prohibitif imposé pour ce genre de traitement. De plus, il est très rare qu'un circuit ne comporte que dix transistors identiques.

3) LES SOLUTIONS ENVISAGEABLES

Pour résoudre ce problème d'explosion numérique, différentes solutions sont envisageables.

D'abord, il faut commencer par l'identification des éléments les moins nombreux. Partant de là, on obtient plus facilement des correspondances entre les noeuds, et on limite donc le nombre de possibilités pour les autres composants, puisque quelques noeuds sont déjà identifiés.

La deuxième méthode à utiliser est liée aux caractéristiques physiques des transistors. En effet, la connexion au substrat d'un transistor est très rarement relié au drain ou à la source d'un autre. De même, une résistance ou une capacité sont très rarement connectées au substrat.

Moyennant ces précautions de recherche, des essais ont montré que le nombre de possibilités pouvait être limité à factorielle N où N est le nombre de composants identiques.

Des méthodes d'identifications heuristiques dérivées de l'intelligence artificielle peuvent être employées pour encore réduire le nombre de recherches. Ces techniques donnent de bons résultats dans le cas où les deux circuits sont identiques. Par contre, en cas d'erreur, il faut explorer toutes les voies possibles afin de donner au concepteur toutes les causes d'erreurs envisageables. Dans ce cas, ces techniques ne sont plus d'une grande utilité.

V- CONCLUSION

Les extractions de schémas ont été testées sur des circuits de petite taille (quelques centaines de transistors).

L'extraction en vue d'une comparaison est très performante et adaptative. En effet, nous pouvons extraire des schémas de circuit en technologie MOS ou bipolaire. Les résultats obtenus sont tout à fait comparables à ceux de DRACULA. Il est à noter que ce produit est limité à la technologie MOS.

L'extraction fine en vue d'une simulation donne elle aussi de bons résultats. Le temps de calcul est plus important (un facteur deux est le minimum), mais les résultats sont très précis. Sur une partie critique comportant environ 20 transistors, les courbes expérimentales et celles obtenues par simulation SPICE différaient de moins de 5%. Nous nous sommes limités à un circuit de 20 transistors, car le nombre de noeuds pouvant être traités par SPICE est limité à une centaine pour des raisons de convergence des calculs. Un circuit de cette taille comporte un peu plus de 100 noeuds générés par l'ajout des résistances, capacités et diodes parasites.

La comparaison de schémas fonctionne sans la mise en oeuvre des modes de détermination heuristiques. Des essais ont été réalisés sur des circuits d'une vingtaine de transistors et les erreurs ont bien été détectées. Le temps de calcul est effectivement très long, d'où la limitation à des circuits de très petite taille. Nous estimons que cette méthode, si le produit fonctionne sur une station de travail, est limitée à des circuits comprenant moins de 50 transistors identiques. En conséquence, pour des circuits de taille importante, il paraît utile d'envisager l'utilisation de simulateur au niveau "switch" permettant de valider des circuits de taille plus importante. L'inconvénient de cette méthode est de ne pas fournir de renseignements sur la localisation des différences, et donc de fournir très peu d'aide au concepteur dans l'identification de l'erreur.

REFERENCES

[1] A. KOESTLER, "The ghost in the machine", Henry REGNERY Co., Chicago, 1967

[2] H. J. SIMON, "The architecture of complexity", Proceeding of the american philosophical society, vol. 106, no. 6, December 1962

[3] W. M. NEWMANET R. F. SPROULL, "Principles of interactive graphics", 2d ed., McGraw-Hill, New York, 1979

[4] J. D. WILLIAMS, "STICKS - A graphical compiler for high-level LSI design", AFIPS conference proceedings, vol. 47, p. 289-295, June 1978

[5] M. Y. HSUEH, "Symbolic layout compaction", computer design aids for VLSI circuits, édité par ANTOGNETTI, PEDERSON et DE MAN, SIJTHOFF an NOORDHOFF, p. 499-541, 1981

[6] H. BEKE, W. M. C. SANSEN et R. VAN OVERSTRAETEN, "CALMOS : A computer-aided layout program for MOS/LSI", IEEE journal of solid-state circuits, vol. SC-12, no. 3, p.281-282 Juin 1977

[7] G. PERSKY, D. N. DEUTSCH et D. G. SCHWEIKERT, "LTX - A minicomputer-based system for automated LSI layout", journal of design automation an fault tolerant computing, vol. 1, no.3, p. 217-255, May 1977

[8] D. L. JOHANSEN, "A design technique for VLSI chips", Proceedings of the Caltech conference on VLSI, January 1979

[9] D. L. JOHANSEN, "Silicon compilation", Technical report no. 4530 - Departement of computer science, California institute of technologiy, Pasadena, 1981

[10] C. W. ROSE, M. ORDY et F. I. PARK, "NmPc: A retrospective", Proceedings ACM/IEEE 20th Design Automation Conference, p. 506-514, Juin 1983

[11] A. GUPTA, "ACE: A circuit extractor", Proceedings ACM/IEEE 20th Design Automation Conference, p. 721-725, Juin 1983

[12] L. W. NAGEL, "SPICE2: A computer program to simulate semiconductor citcuits", ERL memo, ERL-520 University of California, Berkeley, Mai 1975

[13] J. OUSTERHOUT, "Magic : A VLSI layout system", Proceedings ACM/IEE 21st Design Automation Conference, p. 152-159, Juin 1984

[14] C. J. TERMIN, "RSIM- A logic level timing simulator", Proceedings IEEE International Conference on Computer Design, 31 Octobre-3 Novembre 1983

[15] J. SOUKUP, "Circuit Layout", Proceedings IEEE, Vol. 69, no. 10, p. 1281-1304, Octobre 1981

[16] H.S. BAIRD, "Fast algorithms for LSI artwork analysis" Proceedings of 14 th IEEE Design automation Conference, June 1977

[17] P. WILCOX, H. ROMBECK and D.M.CAUGHEY, "Design rule checking based on one dimentional scan" - Proceedings of 15 th Design Automation Conference, June 1978

[18] "Critic : an integrated circuit-design rule checking program", IEE international conference on computer aided design, 1972

[19] D. WOLF ans U. JAEGER, "Traitement de données géométriques pour circuits intégrés" Nachn. Tech., 1978

[20] M. YAMIN, "A computer program for integrated circuit mask design checkout" Belle System technical journal, 1972

[21] K. YOSHIDA, T. MITSUHASHI, Y.NAKADA, T. CHIBA, H.OGITA and S. NAKATSUKA" "A layout checking system for large scale integrated circuit" - Proceedings of 14 th design automation conference, 1977

[22] H.S. BAIRD and Y.E. CHO, "An artwork design verification system" - Proceedings of the 12th design automation conference, 1975

[23] K. KOLLET and U. LAUTHER, "Autopruef... a program for geometrical testing of integrated circuit mask designs"Siemens Forsch. Entwickl., 1973

[24] B.W. LINDSAY and B.T. PREAS, "Design Rule checking and Analysis IC Mask Designs" - Proceedings of 13th Design automation conference

[25] C.L. MITCHELL and J.M GOULD, "MAP : a user-controlled Automated Mask Analysis Program" - Proceedings of the 11th Design automation workshop, 1974

[26] B.T. PREAS, B.W. LINDSAY and C.W. GWYN, "Automatic circuit analysis based on Mask Information" - proceedings of the 13th design automation conference, 1976

[27] M. YAMIN, "XYTOLR... a computer program for integrated circuit mask design checkout" Bell System Technical Journal, 1972

[28] M. YAMIN, "Derivation of all figures formed by the intersection of generalized polygons" Bell System Technical Journal, 1972

[29] H.S. BAIRD, "Design of a family of algorithms for large scale integrated circuit mask artwork analysis"Master's thesis, 1976

[30] C. BAKER and C. TERMAN, "Tools for verifying integrated circuit designs" VLSI Design, 1980

[31] R. LYON, "Simplified design rules for VLSI layouts" VLSI design, 1981

[32] T. WHITNEY, "A hierarchical design rule checking algorithm" VLSI Design, 1981

[33] H.S. BAIRD, "A survey of computer aids for mask artwork verification", - Proceeding of the IEEE International Symposium, 1977

[34] E. J. Mc GRATH and T. WITNEY, "Design Integrity and Immunity chacking : a new look at layout verification and design rule checking" - Proceedings of the 17th design automation conference, 1980

ANNEXES

I– ANNEXE 1 : Règles de conception en technologie NMOS 6 μm

Ce paragraphe présente les informations utiles pour le dessin de circuits intégrés utilisant une technologie de fabrication NMOS grille polysilicium 6 μm.

1) LES MASQUES UTILISES

			Désignation abrégée
Niveau	1	Diffusion	D
	2	Transistors naturels	
	3	Implantation ionique	I
	4	Contact direct	C
	5	Polysilicium	P
	6	Trou de contact	TRC
	7	Surtrou	
	8	Aluminium	M
	9	Passivation	

Les niveaux 2, 4 et 9 sont optionnels.

2) LES REGLES DE GARDE

a– Les connexions

Figure 71 : **Diffusion** VERIF d 6 6

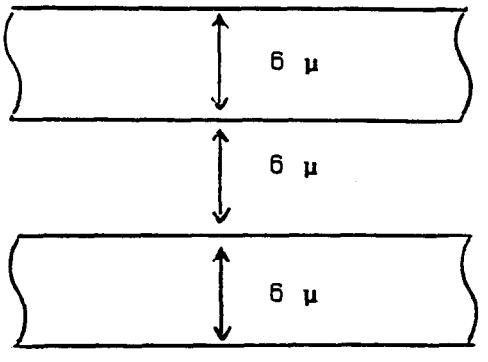


Figure 72 : **Polysilicium** VERIF p 6 6

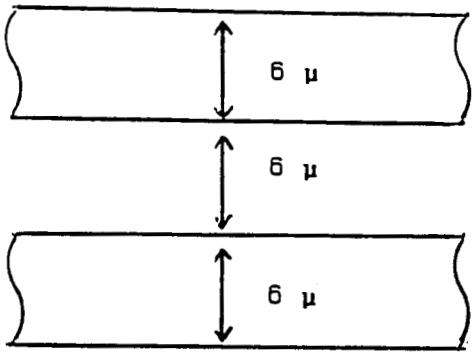


Figure 73 : **Aluminium** VERIF m 10 8

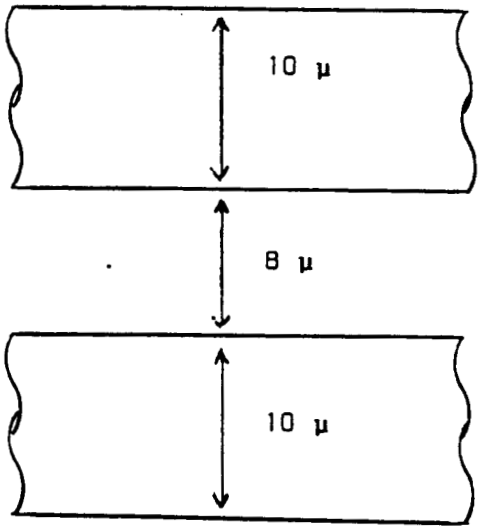
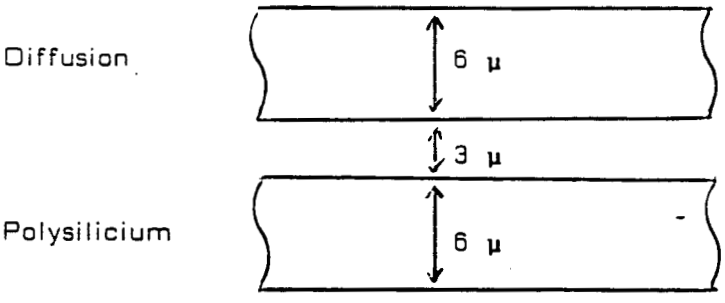


Figure 74 : Distance Diffusion-Polysilicium

cont = c OU trc

SPACING p d 3 (TOUCH CASE cont



b- Les transistors

Figure 75 :

Dépassement de 4 μm du polysilicium au-delà de la diffusion

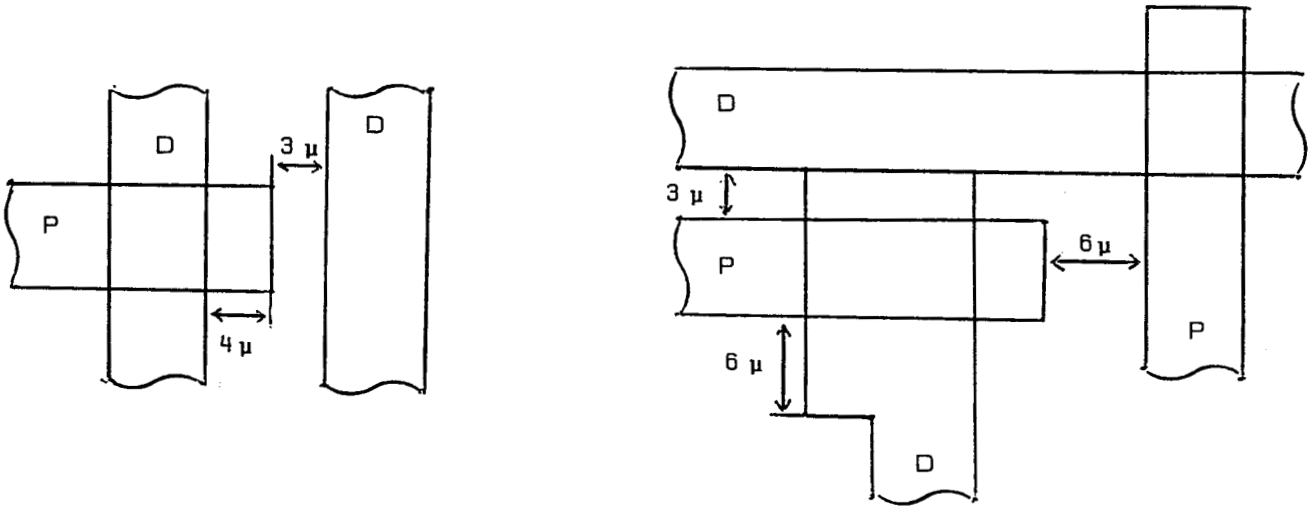


Figure 76 :

a = p ET d
cd = c ET a
a = a MOINS cd
cb = a ET trc
tr = a MOINS cb
VERITR tr p d 4 6
VERIF tr 0 6

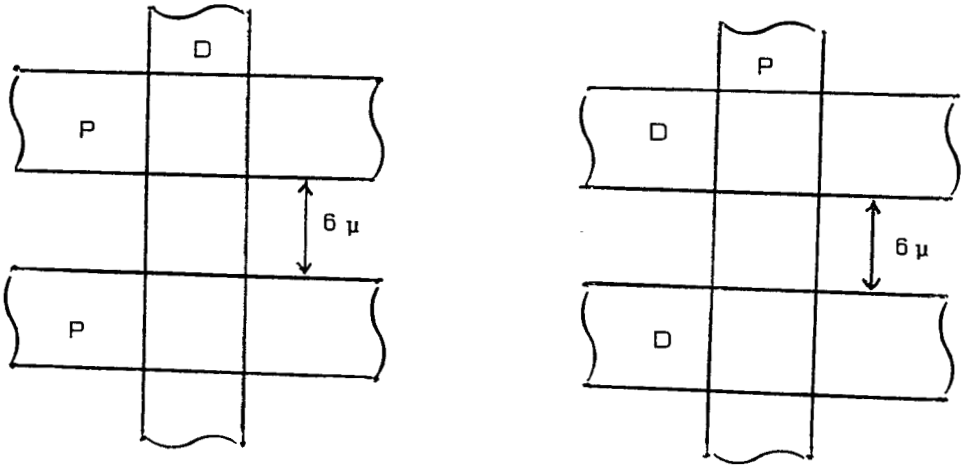
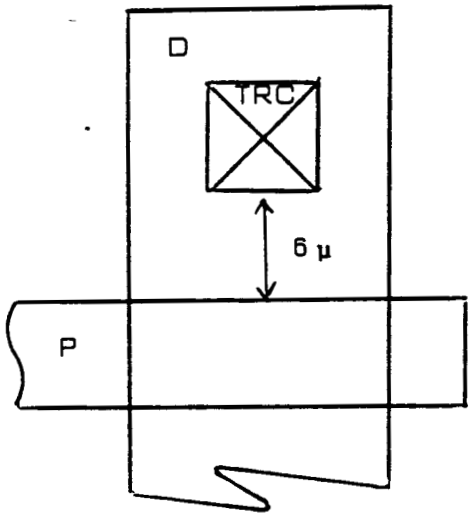


Figure 77 :

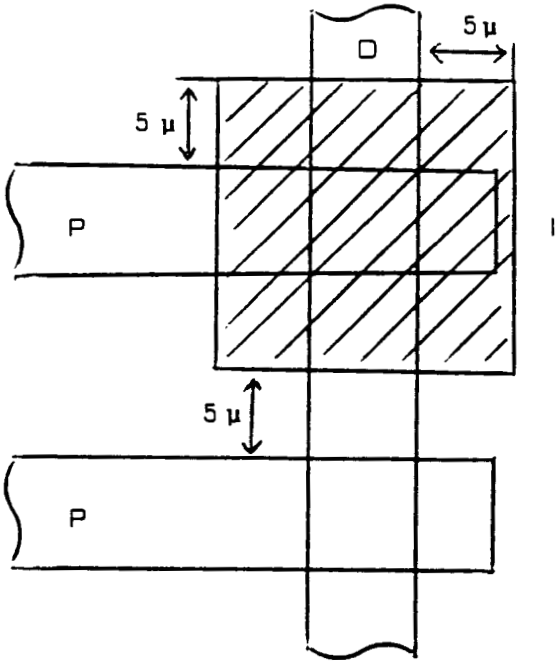
Distance "trou-transistor" de 6 μm



SPACING tr trc 6

Débordement du masque d'implantation (niveaux 2 et 3) de 5 μm autour du canal. Espacement de 5 μm par rapport à un transistor à enrichissement.

Figure 78 :



i COVER tr 5

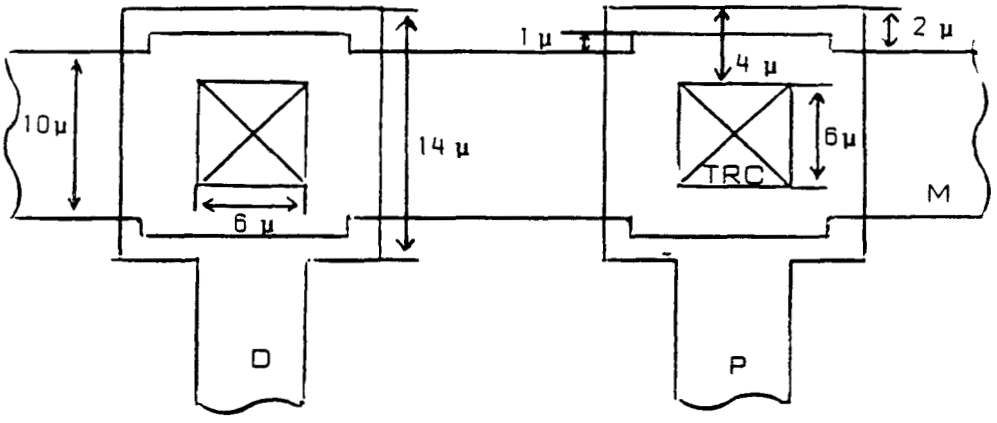
SPACING i tr 5

c– Les trous

Un trou se compose d'un trou de contact (niveau 6) qui est un carré de 6 x 6 μm entouré d'un surtrou (niveau 7) qui est un carré de 10 x 10 μm. D'une manière générale, le niveau 7 peut être obtenu automatiquement à partir du niveau 6 (+ 2 μm). On prend la convention de ne plus dessiner le niveau 7.

Débordement du polysilicium ou de la diffusion de 4 μm autour du trou.
Débordement de 3 μm de l'aluminium autour du trou.

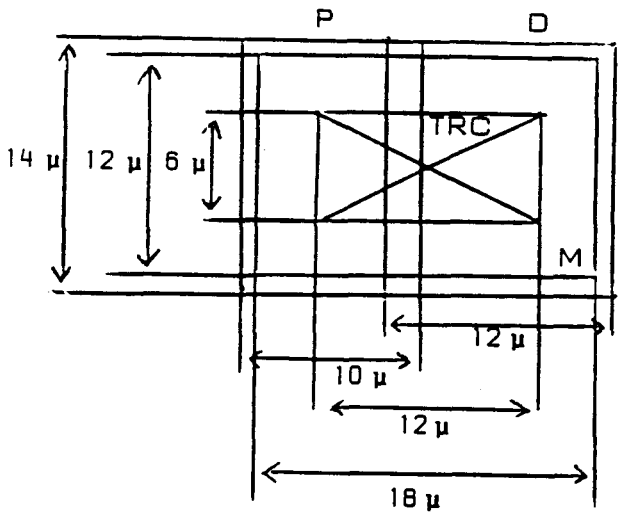
Figure 79 :




```
trcb = MEET trc cb
EQ trcb 6 12
trcn = trc MOINS trcb
EQ trcn 6 6
m COVER trc 3 ( ALL
p COVER trcn 4
d COVER trcn 4
a = trc MOINS p
a = a MOINS d
ERROR a
```

Pour le contact en bout (diffusion, polysilicium, aluminium), le trou doit avoir des dimensions de 6 x 12 μm. Les débordements d'aluminium, diffusion et polysilicium restent identiques.

Figure 80 :

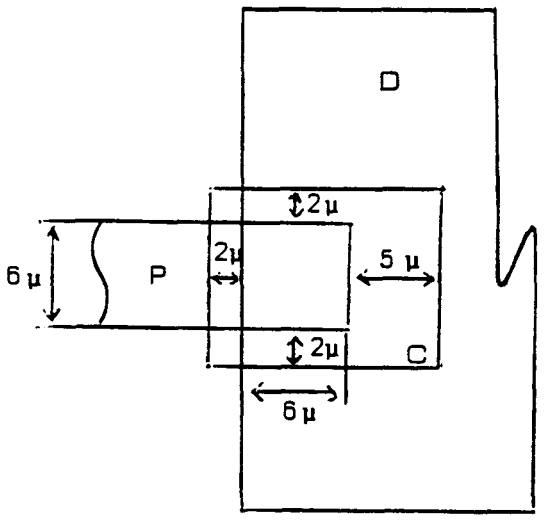


trcbpl + trcb PLUS 4
dcb = d ET trcbpl
pcb = p ET trcbpl
dcb = dcb MOINS trcb
pcb = pcb MOINS trcb
VERIF dcb 4 0
VERIF pcb 4 0

d– Les contacts directs

Le contact direct est un contact polysilicum–diffusion. Le débordement minimum de l'ouverture de contact par rapport à la surface active est défini comme suit :

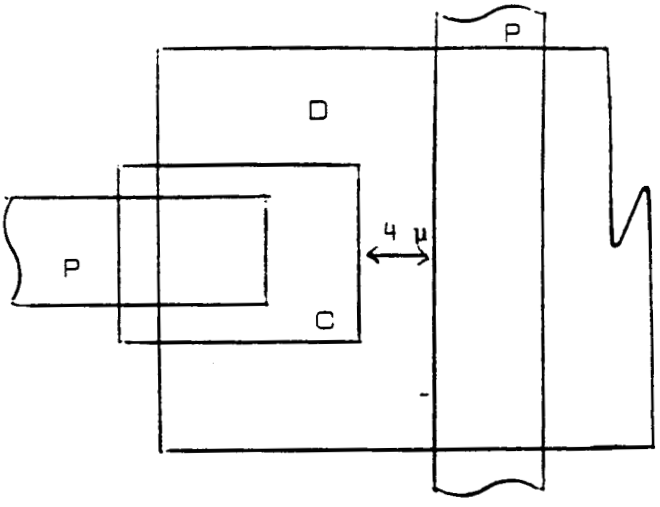
Figure 81 :



VERIF cd 6 0
c COVER cd 2
DIRECT c 2 5

Distance minimum de 4 μm entre un contact direct et une grille active. La règle reste la même que le transistor soit à enrichissement ou à déplétion.

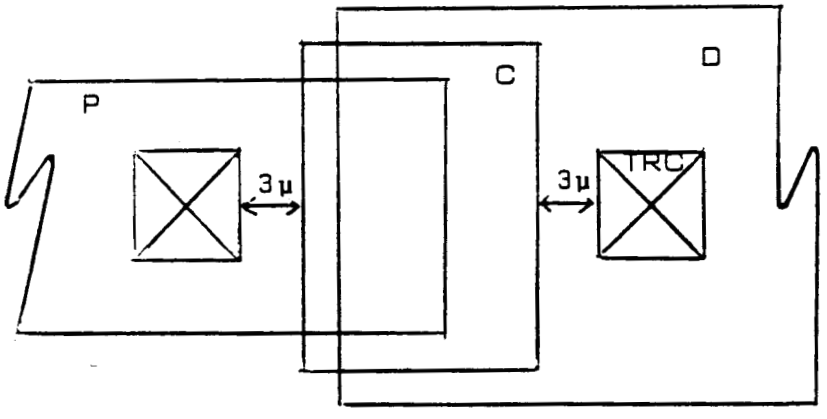
Figure 82 :



SPACING c tr 4

Distance minimum de 3 μm entre un contact direct et un autre contact.

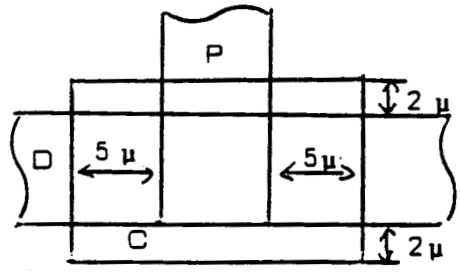
Figure 83 :



SPACING c trc 3

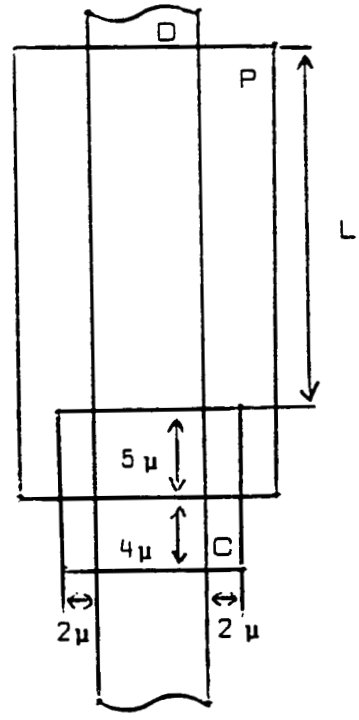
Pour les contacts perpendiculaires, la règle particulière suivante a pour but d'éviter la création d'un transistor MOS parasite.

Figure 84 :



Cas d'un transistor MOS de charge déplété long (longueur du canal supérieure à 20 μm) et de valeur non critique.

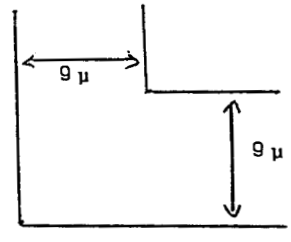
Figure 85 :



e- La passivation

Trou à 9 μm des bords du plot de contact.

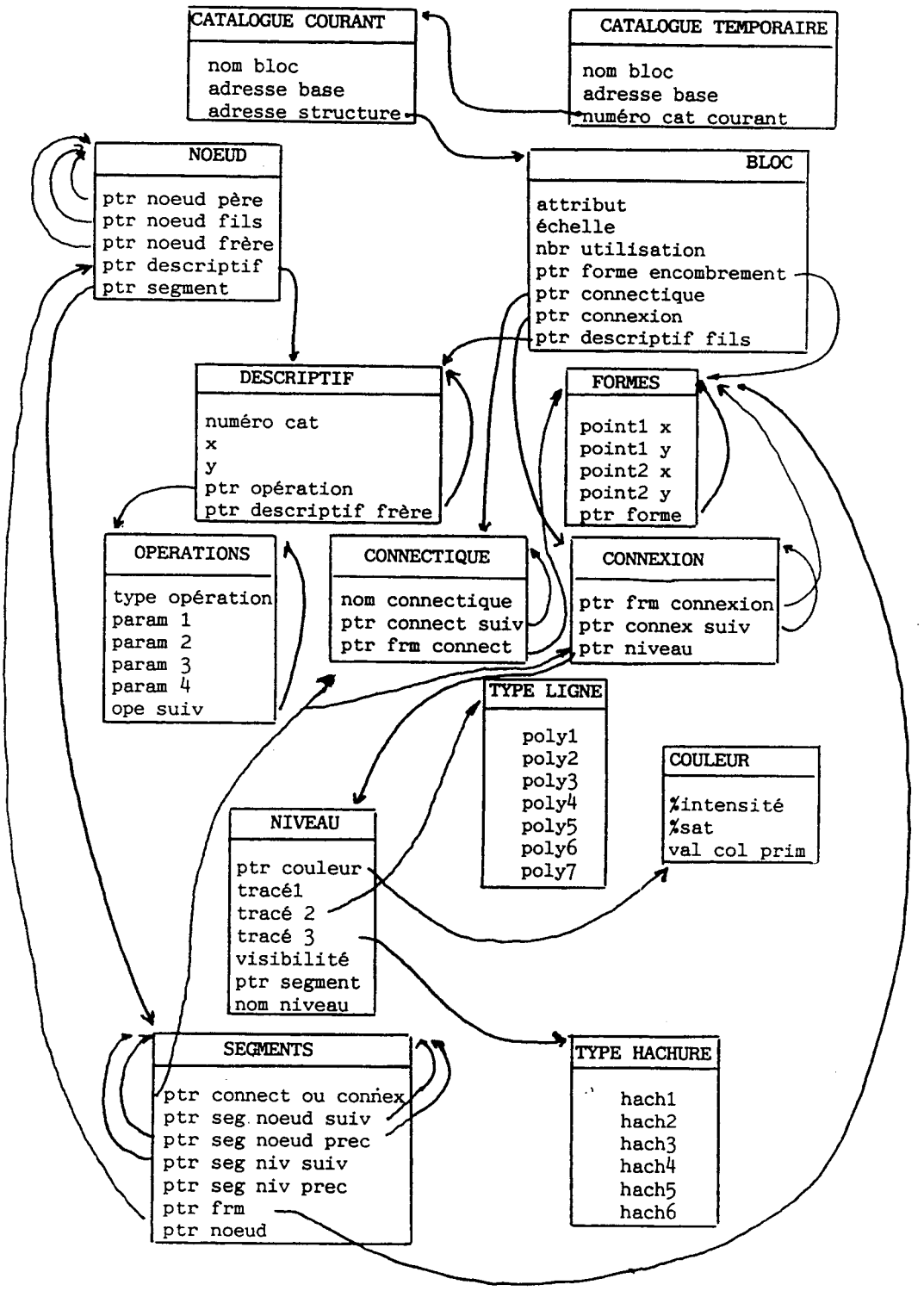
Figure 86 :



II- ANNEXE 2 : La structure de données

1) SCHEMA GENERAL

Figure 87 :



2) LE CATALOGUE COURANT

Les circuits réalisés sont stockés dans des bases de données. Pour chacune d'entre elles, il existe une directory dans laquelle on trouve l'ensemble des circuits mémorisés (cf description de la base). Lors de l'activation de l'éditeur, on ouvre deux bases :

- La base courante qui est la première base accessible en écriture dans le catalogue des bases.

- La base temporaire qui est l'une quelconque des bases du catalogue des bases.

Lors de l'ouverture de la base courante, on charge la directory dans le catalogue courant. Celui-ci est un tableau, chaque ligne contient les informations suivantes :

- NOM BLOC : nom des blocs présents dans la base ou dans la structure (cas d'un bloc en création et qui n'a pas encore été sauvegardé).

- ADRESSE BASE : adresse du bloc dans la base. Cette donnée est nulle dans le cas d'un nouveau bloc, et négative (valeur absolue = adresse base) dans le cas où le bloc a été modifié depuis son chargement. Il faudra dans les deux cas sauvegarder le bloc si l'utilisateur en fait la demande.

- ADRESSE STRUCTURE : adresse du bloc dans la structure de donnée interne lorsque celui-ci a été chargé. Ce pointeur permet de retrouver toutes les informations relatives à un bloc.

Le chargement du catalogue se fait lors de l'ouverture de la base de manière séquentielle dans des positions contiguës. Le premier emplacement libre est connu. Lors des restitutions, l'emplacement libéré est chaîné en tête, le chaînage se fait à l'aide de l'adresse base. Quand on libère un emplacement, celui-ci devient le premier emplacement libre et pointe sur le suivant qui était le premier avant la restitution.

Après chargement d'un bloc, la base courante ne peut plus être changée, par sécurité.

3) LE CATALOGUE TEMPORAIRE

La base temporaire est utilisée, par le concepteur, pour aller rechercher et charger des blocs définis dans d'autres bases. Le catalogue de cette deuxième base est chargé dans un catalogue temporaire. On ne peut accéder à une base temporaire qu'en lecture. Après chargement d'un bloc, cette base temporaire peut être changée de façon à permettre au concepteur d'aller rechercher des blocs de diverses origines.

Le catalogue temporaire contient les informations suivantes :

- NOM BLOC qui sont les noms des blocs définis dans la base temporaire.

- ADRESSE BASE qui est un pointeur désignant l'emplacement du bloc concerné dans la base temporaire.

- NUMERO CAT COURANT est un pointeur qui désigne l'emplacement du bloc dans le catalogue courant si celui-ci a été chargé dans le bloc en cours d'édition. Si le bloc n'a pas été préalablement chargé, ce pointeur est nul. La présence d'un pointeur positif indiquera que le bloc concerné a été chargé pour l'édition du circuit en cours et que celui-ci ne doit plus être chargé pour une utilisation ultérieure.

Ce catalogue ne peut en aucun cas évoluer puisque la base temporaire n'est accessible qu'en lecture.

4) LES BLOCS

Le bloc constitue l'entité qui permet de retrouver toutes les informations de type géométrique qui vont permettre de reconstituer une image graphique du bloc. Le bloc est défini par un encombrement, des connectiques et des connexions. Chacun de ces éléments est matérialisé géométriquement parlant par des figures (polygones, lignes, rectangles, etc...). Chacune de ces figures sera stockée dans des formes (ensemble de deux points).

Un bloc est donc décrit par les éléments suivants :

- ATTRIBUT : c'est un indicateur qui permettra de préciser les droits de modification du bloc. Certains blocs ne pourront être accessibles qu'en lecture.

- ECHELLE : toutes les coordonnées graphiques utilisées dans ELODIE sont des multiples entiers d'un pas (ex le 10 ème de μm). La valeur de ce pas est donnée par l'échelle.

- NBR D'UTILISATION : pour ne pas encombrer inutilement la mémoire, il convient de compter le nombre de fois où un bloc est utilisé. On pourra ainsi sortir le bloc de la structure de données dès que celui-ci n'est plus référencé.

- PTR FORME ENCOMBREMENT : pointeur de formes de l'encombrement. Il désigne la première forme utilisée pour décrire le ou les polygones de l'encombrement du bloc.

- PTR CONNECTIQUE : pointeur de connectique. Ce pointeur désigne la première connectique du bloc considéré.

- PTR CONNEXION : pointeur de connexion. Ce pointeur désigne la première connexion du bloc.

- PTR DESCRIPTIF FILS : pointeur descripteur fils. Ce pointeur désigne le descriptif du premier fils du bloc considéré.

5) LES DESCRIPTIFS

Les descriptifs permettent de décrire les liens de dépendance père fils. Tout bloc doit être pointé par un descriptif même si celui-ci est la racine de l'arborescence.

Chaque descriptif contient les informations suivantes :

- NUMERO CAT : Il s'agit d'un pointeur dans le catalogue courant ou temporaire. On ne désigne pas le bloc directement dans la structure. La distinction entre le catalogue courant et le catalogue temporaire se fait en fonction du signe du pointeur.

- X et Y : coordonnées du point origine du bloc fils par rapport aux coordonnées origines du bloc père. Les coordonnées sont données avec l'échelle du bloc englobant.

- PTR OPERATION : pointeur vers la première opération subie par le bloc fils au cours de l'insertion dans le bloc père.

- PTR DESCRIPTIF FRERE : ce pointeur permet de désigner le descriptif d'un bloc frère. Un bloc peut en effet disposer de plusieurs fils. Le premier descripteur fils est atteint à partir du bloc père, les autres descripteurs sont chaînés les uns aux autres. L'utilisation du pointeur descriptif fils et de ce pointeur permet de parcourir l'arborescence des dépendances entre blocs.

6) LES OPERATIONS

Un bloc peut subir des opérations : rotation, symétrie /x, symétrie /y, répétition. Les paramètres relatifs à une opération sont rangés dans une ligne du tableau opération.

- TYPE OPERATION : Il précise le type d'opération que le bloc a subi
  1 = rotation
  2 = symétrie /x
  3 = symétrie /y
  4 = répétition

- PARAMi : ce sont les paramètres nécessaires pour définir complètement l'opération :

  * les symétries n'utilisent aucun paramètre.

  * Pour les rotations, un seul paramètre intervient, l'angle de rotation en degré. Le centre de rotation est le point origine du bloc. Pour toutes les rotations le sens trigonométrique a été retenu.

  PARAM1 nombre de répétitions suivant l'axe x
  PARAM2 distance entre répétitions sur l'axe x
  PARAM3 nombre de répétitions suivant l'axe y
  PARAM4 distance entre répétitions sur l'axe y

- OPE SUIV : un bloc peut subir différentes opérations. Celles ci sont chaînées les unes aux autres par ce pointeur.

7) LES FORMES

L'encombrement, les connectiques, les connexions sont schématisées par des figures : polygone, rectangle, ligne etc... Il faut stocker les différentes coordonnées des points qui constituent la figure :

- pour un rectangle, les deux points de la diagonale

- pour un polygone, les points du contour

- pour une ligne, les points de la ligne.

Une succession de points ne suffit donc pas à définir la figure, il faudra ajouter un qualificateur.

La mémorisation des figures se fait sous la forme de couples de points. Les différents couples sont chaînés les uns aux autres par un pointeur qui assure à la fois le chaînage et la qualification de la figure.

Une connexion, une connectique, un encombrement peuvent être constitués de plusieurs figures. Ces figures sont chaînées les unes aux autres par le pointeur de chaînage précédent. La distinction de la signification du chaînage se fait en jouant sur le signe.

- POINT1 X POINT 1 Y : coordonnées du premier point.

- POINT2 X POINT2 Y : coordonnées du deuxième point.

- PTR FORME : pointeur vers la forme ou la figure suivante.

  * si le pointeur est négatif, ceci indique que la figure n'est pas terminée. La valeur absolue du pointeur désigne la forme suivante.

  * si le pointeur est positif, ceci indique que la figure est terminée. Le pointeur sert alors de lien de chaînage et de qualificateur de formes.

* si le pointeur est strictement supérieur à zéro et strictement inférieur à BAFR1, la figure est un polygone. Le pointeur désigne la figure suivante.

* si le pointeur est strictement supérieur à BAFR1 et strictement inférieur à 2*BAFR1, la figure est une ligne. En retirant BAFR1 au pointeur, on pointe la figure suivante.

* si le pointeur est strictement supérieur à 2*BAFR1 et strictement inférieur à 3*BAFR1, la figure est un rectangle. En retirant 2*BAFR1 au pointeur on pointe la figure suivante.

* si le pointeur est égal à 0 ou BAFR1 ou 2*BAFR1, la figure est respectivement un polygone , une ligne ou un rectangle et que celle-ci est la dernière figure chaînée.

Les coordonnées rangées dans le tableau forme sont réelles bien que celles-ci soient des multiples entiers du pas défini pour le bloc. Dans le cas où un seul point est défini dans la forme, ses coordonnées sont dupliquées dans le deuxième point.

8) LES CONNEXIONS

Les connexions décrivent les parties les plus internes du bloc lorsque celui-ci n'a pas de descendance. Dans le cas contraire, elles décriront les interconnexions entre les fils du bloc auquel elles sont associées. A chaque connexion est associé un niveau (ex : métal, polysilicium) qui permettra de caractériser le graphisme associé.

Une connexion comporte les éléments suivants :

- PTR FRM CONNEXION : pointeur vers la première forme de la connexion.

- PTR CONNEX SUIV : pointeur vers la connexion suivante. Il y a une connexion par niveau utilisé et l'ensemble des connexions d'un bloc sont chaînées.

- PTR NIVEAU : pointeur vers le niveau associé à la connexion, c'est à dire vers les éléments qui caractériseront le graphisme.

9) LES CONNECTIQUES

Les connectiques décrivent les points d'entrée et de sortie d'un bloc. Elles sont schématisées par une forme. Electriquement parlant, elles n'ont pas d'existence réelle.

Une connectique comporte les éléments suivants :

- NOM CONNECTIQUE : il contient le nom de la connectique (ex VCC....).

- PTR CONNECT SUIV : pointeur vers la connectique suivante du bloc.

- PTR FRM CONNEC : pointeur forme connectique. Il pointe sur la première forme de la figure qui schématise la connectique.

10) LES NOEUDS

Le noeud permet de spécifier de façon précise un bloc au niveau d'un schéma.

Un bloc peut être utilisé plusieurs fois dans un autre bloc. Dans ce cas sa descendance (descriptifs) n'est pas dupliquée. Cependant il faut pouvoir désigner de façon précise un bloc de la descendance. On décrit l'ensemble des dépendances Père Fils grâce à l'arborescence des noeuds.

Un noeud est décrit de la manière suivante :

- PTR NOEUD PERE, PTR NOEUD FILS, PTR NOEUD FRERE : pointeur de description de l'arborescence respectivement vers le père, le premier fils et le premier frère.

- PTR DESCRIPTIF : pointeur vers le descriptif du bloc associé au noeud.

- PTR SEGMENT : pointeur vers le premier segment du noeud.

11) LES SEGMENTS

Le segment n'a d'existence qu'au niveau graphique. La visualisation de tout ou partie d'un bloc s'effectuera en faisant appel à des primitives de tracé de la bibliothèque graphique GKS.

Un segment graphique est composé d'un ensemble de tracés élémentaires. Cet ensemble constitue un tout qui peut être manipulé par des opérations de rotation, translation, par des commandes de visibilité ou de détectabilité etc... Un segment est désigné par un numéro compris entre 0 et 32767.

Un segment est décrit par les éléments suivants :

- PTR CONNECT OU CONNEX : pointeur connectique ou connexion. Ceci permet de savoir quel élément du bloc est concerné par le segment.

si le pointeur est positif, il s'agit d'une connexion.

si le pointeur est négatif, il s'agit d'une connectique.

si le pointeur est nul, il s'agit de l'encombrement. Dans ce dernier cas, on peut retrouver la forme associée par le bloc.

- PTR SEG NOEUD SUIV, PTR SEG NOEUD PREC : tous les segments d'un noeud sont chaînés les uns aux autres sous forme d'une liste bilatère afin de pouvoir réaliser rapidement des opérations sur tous les segments d'un même noeud. Ces deux pointeurs désignent respectivement le segment suivant et le segment précédent.

- PTR SEG NIV SUIV, PTR SEG NIV PREC : tous les segments d'un même niveau sont chaînés les uns aux autres sous la forme d'une liste bilatère afin de pouvoir réaliser rapidement des opérations sur tous les segments d'un même niveau (ex : visibilité d'un niveau). Ces deux pointeurs désignent respectivement le segment suivant et le segment précédent.

- PTR FRM : ce pointeur désigne la forme associée au segment. Dans l'environnement de visualisation, un segment contient plusieurs formes, on génère un segment par niveau. Le pointeur désigne alors la première forme. Dans l'environnement d'édition, chaque segment représente une forme élémentaire.

- PTR NOEUD : Ce pointeur désigne le noeud auquel appartient le segment. C'est par ce chemin qu'on peut atteindre la forme associée à l'encombrement.

12) LES NIVEAUX

Les niveaux permettent de caractériser les graphismes utilisables pour décrire le circuit.

Un niveau comporte les éléments suivants :

- PTR COULEUR : pointeur dans la table des couleurs.

- TRACE1 : indicateur précisant s'il s'agit d'un tracé de type ligne ou de type polygone plein.

- TRACE2 : si le tracé est une ligne , il s'agit d'un pointeur dans la table des types de trait (7 possibles). Si le tracé est un polygone plein, il précise le type de remplissage (vide, plein, hachuré).

- TRACE3 : si le tracé est un polygone plein il s'agit d'un pointeur dans la table des types de hachures dans le cas d'un polygone plein hachuré (6 possibles).

- VISIBILITE : cet indicateur précise si le niveau doit être visible ou non.

- PTR SEGMENT : pointeur vers le premier segment associé au niveau.

- NOM NIVEAU : nom associé au niveau.

13) LES TYPES DE HACHURES

Cette table contient les paramètres à fournir aux fonctions GKS pour préciser le type de hachure. Six types de hachures peuvent être définis. Le graphisme obtenu dépend du terminal utilisé.

14) LES TYPES DE LIGNES

Cette table contient les paramètres à fournir aux fonctions GKS pour préciser le type de ligne . Sept types de lignes peuvent être définis. Le graphisme obtenu dépend du terminal utilisé.

15) LES COULEURS

Chaque couleur est définie à l'aide de trois paramètres :

- %INTENSITE : valeur de l'intensité.
- %SATURATION : valeur de la saturation.
- VAL COL PRIM : valeur de la couleur primaire comprise entre 0 et 6.

A partir de ces trois valeurs, on calcule les pourcentages de rouge, de vert et de bleu. On peut définir au maximum 256 couleurs.

III- ANNEXE 3 : Structure de la base de données

1) STRUCTURE FONCTIONNELLE

Voici la structure fonctionnelle des différentes entités stockées dans la base.

- Les BLOCS

/Nom bloc/Attribut/Echelle/Pointeur/Pointeur/Pointeur/Pointeur/
/ / / / figure / connect/ fils / opérat /

- Les FORMES

/FORM1 Q1 NIV1/..../FORMn Qn NIVn/

- Les CONNECTIQUES

/CON1 N1/CON2 N2/..../CONm Nm/

- Les DESCRIPTIFS

/FILS1X1 Y1 NOP1/..../FILSk Xk Yk NOPk/

- Les OPERATIONS

/ OP1 / OP2 /..../ OPp /

a- Informations générales relatives au bloc

nom du bloc	8 caractères
attribut	entier
échelle	réel
pointeur figure	entier -> n° enregistrement
pointeur connectique	entier -> n° enregistrement
pointeur fils	entier -> n° enregistrement
pointeur opération	entier -> n° enregistrement
	--------
	32 octets

b- Les figures

Une figure est un ensemble de formes. Une forme est un ensemble de 2 points suivi d'un qualificateur Q :

. Q < 0 la forme n'est pas la dernière de la figure
. Q = 0 fin d'une figure de type polygone
. Q = BAFR1 fin d'une figure de type ligne
. Q = BAFR2 fin d'une figure de type rectangle

On y ajoute le niveau auquel appartient la figure :

. NIV = 1 -> connectique
. NIV = 2 -> encombrement
. NIV = 3 -> niveau de connexion

Soit un total de 20 octets/formes

c- Les connectiques

On y range les noms des connectiques (8 caractères), auquel on ajoute le nombre de figures appartenant à chaque connectique (entier sur 2 octets). Ce comptage permettra de réassocier chaque figure à chaque connectique. On suppose que le rangement et la lecture se font dans l'ordre des noms. Ce qui nous donne 10 octets/connectique.

d- Description de la descendance

La représentation de la descendance d'un bloc restera très voisine de la notion de descriptif rencontrée dans l'étude de la structure, mais on mémorisera ici le nombre d'opérations que le bloc a subi, en plus de sa position par rapport au bloc père. Soit 16 octets/opération.

e- Les opérations

On y rangera les cinq paramètres relatifs à une opération :

. type d'opération
. param1
. param2
. param3
. param4

2) IMPLEMENTATION PHYSIQUE

Hormis la partie contenant les informations générales du bloc qui sera unique quel que soit le bloc, on ne peut prévoir le nombre d'informations des autres types, qui peut varier suivant la complexité des blocs ou les préférences du concepteur. On va donc regrouper ces entités élémentaires sous forme d'enregistrement, et chaîner ces enregistrements entre eux.

a- Longueur des enregistrements

Un récapitulatif des résultats précédents :

- Info. générales	32 octets
- Forme	20 octets
- Connectique	10 octets
- Fils	16 octets
- Opération	20 octets

nous permet de choisir une longueur d'enregistrement de 80 octets à laquelle il faut ajouter quatre octets destinés au chaînage des enregistrements de même type.

b- Architecture générale

Une base sera implantée sous forme de fichier en accès direct. D'une façon générale ce fichier peut être décomposé en quatre sous-ensembles :

. L'entête de la base. C'est le premier enregistrement de la base, il contient des valeurs décrivant la base.

. La directory ou catalogue de la base. Elle est constituée d'un ensemble d'enregistrements chaînés, contenant chacun les informations générales de 2 blocs.

. Une partie descriptive de blocs, contenant les autres types d'enregistrements sous forme chaînée.

. Une zone libre constituée d'enregistrements non occupés. Ceux-ci sont gérés sous forme de liste chaînée et de liste contiguë.

c- L'entête de base

Elle contient une description physique de la base :

. Longueur de la base (nb d'enregistrements)

. Nombre d'enregistrements utilisés

. Pointeur vers le premier enregistrement de la directory

. Pointeur vers le premier enregistrement de l'espace libre (chaîné ou contiguë)

. Nombre de blocs définis dans la base

d- La directory

Elle est constituée d'enregistrements chaînés. Chaque enregistrement contient les informations générales de 2 blocs.

Le pointeur enregistrement suivant est positionné à -1 pour indiquer la fin du chaînage.

e) Format des enregistrements

1) Enregistrements de formes

Chaque forme correspond aux coordonnées de 2 points soit 16 octets et un qualificateur de forme + niveau

Définition du qualificateur de forme : entier permettant de déterminer si la forme est la dernière d'une figure, si non alors Q = 0, si oui alors Q sera la somme du niveau auquel la figure est attachée et d'un identificateur de fin de forme (0 pour un polygone, BAFR1 pour une ligne, BAFR2 pour un rectangle).

Structure de l'enregistrement

$FORM^i$	X1	Y1	X2	Y2	Q^i
Ptr suiv.	$FORM^i$	$FORM^{i+1}$	$FORM^{i+2}$	$FORM^{i+3}$	

2) Enregistrements de connectiques

On y retrouve les noms des connectiques du bloc. Pour chacune d'elle on mémorise le nom (8 caractères) et le nombre de figures correspondant à cette entité. On place 8 connectiques par enregistrement.

3) Enregistrements des fils

Ils permettent de décrire la descendance d'un bloc, donc pour chaque fils :

. Un pointeur vers le bloc fils

. Un point origine relatif au bloc père

. Le nombre d'opérations subi par le bloc fils

On peut ainsi ranger 5 fils par enregistrement

Pour pointer le bloc fils, on multiplie par 2 le pointeur d'enregistrement auquel on ajoute respectivement 0 ou 1 suivant que le bloc est le 1er ou le second de l'enregistrement.

4) Enregistrements d'opérations

Il y a cinq paramètres à conserver par opération, on peut donc ranger quatre opérations par enregistrement.

Tout comme les autres types d'enregistrements, les enregistrements d'opérations seront chaînés jusqu'à la rencontre d'un pointeur négatif qui indiquera la fin de la chaîne.

f- Espace libre dans la base

Tous les enregistrements inoccupés de la base sont gérés sous forme d'espace libre. Initialement, cet espace correspond à toute la base (c'est alors un espace contiguë). Le pointeur successeur est toujours positionné à 0. Lors des restitutions on se contente de chaîner les enregistrements les uns aux autres, le dernier pointant sur le premier enregistrement de l'espace contiguë.

g- Exemple de bibliothèque

La figure suivante montre l'implémentation d'un bloc avec :

- 2 figures d'encombrement
- 3 connectiques (1ère=2 fig, 2nde=1 fig, 3ème=2 fig)
- 10 figures de connexions
- 4 fils

01	20	9 2 14 5
02	-1	Bloc:Ptr fig=11:Ptr con=3:Ptr fil=8:Ptr ope=5
03	-3	Connectiques
04	18	Formes
05	-2	Opérations
06	13	Formes
07	09	
08	-4	Fils
09	19	
10	07	
11	06	Formes
12	10	
13	04	Formes
14	15	
15	16	
16	17	
17	12	
18	-1	Formes
19	00	
20	00	

IV– ANNEXE 4 : Les fonctions disponibles

1) L'ENVIRONNEMENT D'ACCUEIL

Cet environnement propose les fonctions suivantes :

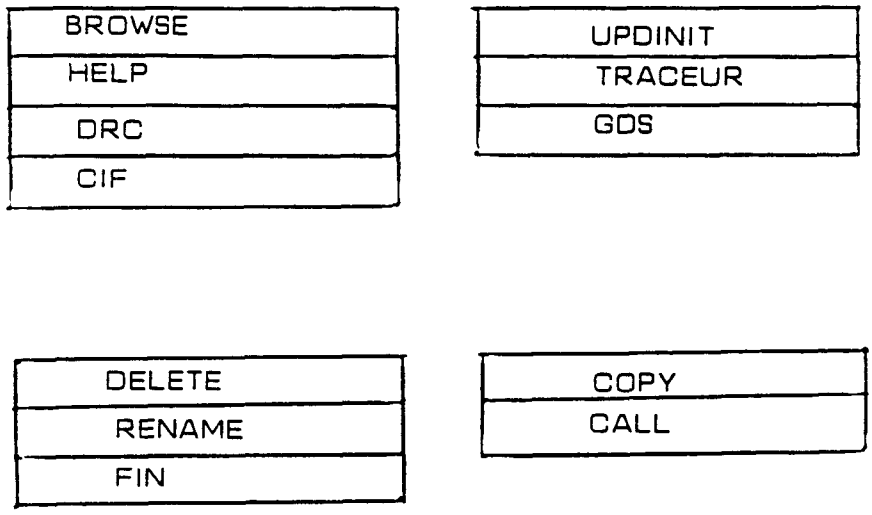
- DELETE : effacer un bloc dans la base
- COPY : copier un bloc de base à base
- RENAME : renommer un bloc dans une base
- CALL : appeler un circuit dans la base, le charger en mémoire, afficher le premier niveau de hiérarchie et le menu2
- FIN : sortie du programme
- BROWSE : visualisation d'un circuit de la base sans possibilité de modification
- UPDINIT : mise à jour du fichier d'initialisation
- HELP : informations sur les bases et les circuits
- TRACEUR : envoi sur table traçante de tout ou partie du circuit

- DRC : appel du vérificateur de garde

- GDS : chargement ou création d'un fichier en format GDS2

- CIF : idem pour le format CIF

Figure 88 :



2) ENVIRONNEMENT DE VISUALISATION/PLACEMENT DE CIRCUIT/BLOC

Cet environnement propose les fonctions suivantes :

- FSCREEN : sélection d'un bloc pour affichage plein écran

- TOP : retour au niveau un de la hiérarchie

- DOWN : affichage de un ou plusieurs niveaux de hiérarchie supplémentaires

- UP : remontée de un ou plusieurs niveaux dans la hérarchie

- ZOOM : sélection d'une partie du circuit pour affichage plein écran (déplacement de cette fenêtre sur le circuit par sélection du sens de déplacement sur une icône constituée d'un ensemble de flèches et d'un cercle où se trouve indiqué si le déplacement est de la taille d'une fenêtre ou d'une demi fenêtre)

- GET : appel d'un bloc dans la base pour insertion dans le circuit en édition

- MOVE : déplacement d'un bloc sélectionné

- COPY : duplication d'un bloc

- ROTATION : rotation d'un bloc sélectionné

- SYMETRIE : symétrie en X ou Y d'un bloc

- REPETER : répétition matricielle d'un bloc sélectionné

- VIEW : visualisation de tout le circuit dans une petite fenêtre (utile pour le zoom)

- VDRC : visualisation des erreurs détectées par le vérificateur

- SAVE : sauvegarde dans la base sans quitter l'environnement d'édition

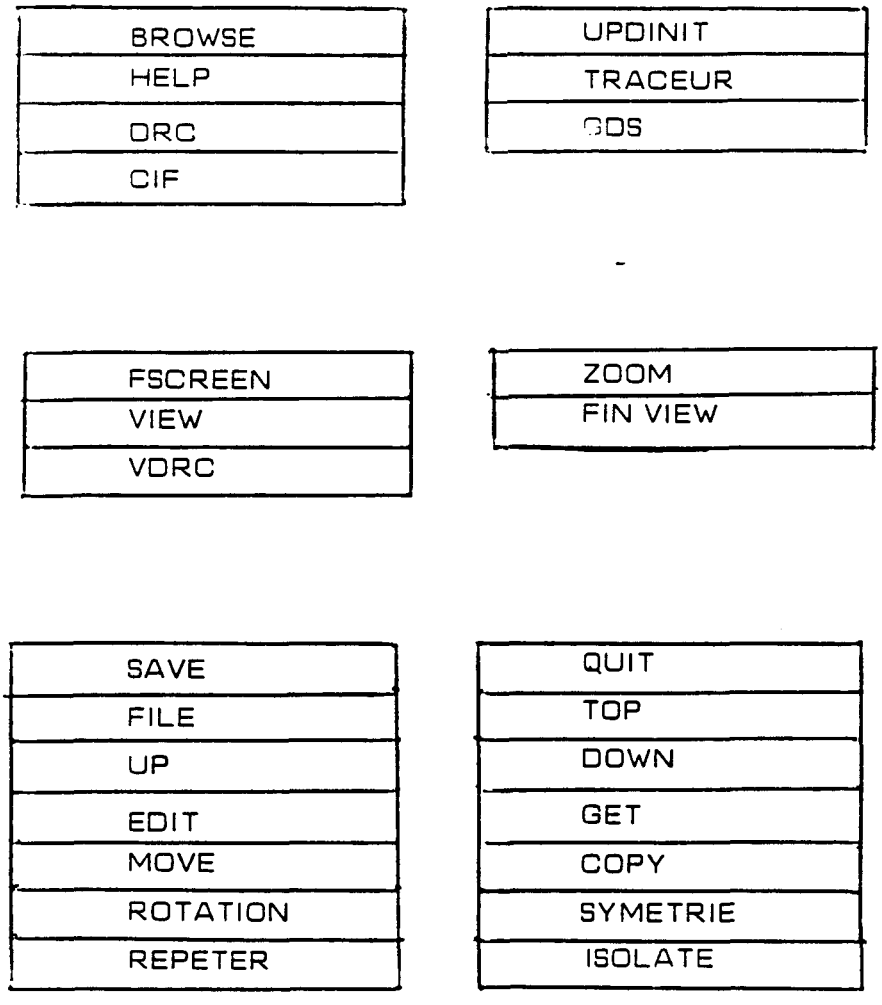
- QUIT : sortie sans sauvegarde

- FILE : équivaut à SAVE puis QUIT

- EDIT : passage au menu 3 sur un bloc

- ISOLATE : individualisation d'un bloc (un élément de matrice par exemple) pour personnalisation. Cette fonction suppose une mise à plat partielle de la structure

Figure 89 :



3) ENVIRONNEMENT D'EDITION DE BLOC

Cet environnement propose les fonctions suivantes :

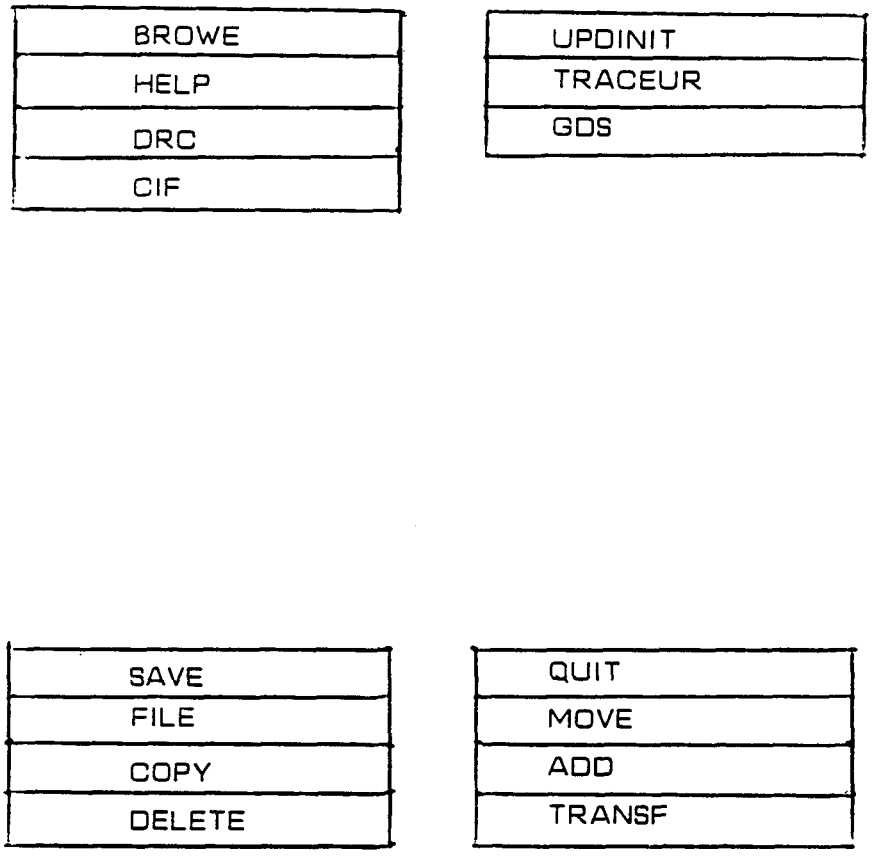
- MOVE : déplacement d'une entité (rectangle, polygone, ligne, text)
- COPY : copie d'un élément
- TRANSF : transformation d'une forme (rectangle en polygone...)

- ADD : ajout d'une forme

- DELETE : destruction d'un élément

- QUIT, SAVE, FILE : sortie avec ou sans sauvegarde

Figure 90 :



V- ANNEXE 5 : Présentation de DRACULA et UDRC

1) DRACULA DE ECAD

Le produit DRACULA est constitué d'un ensemble de logiciels permettant la vérification et la validation d'un circuit. Ces différentes composantes sont :

- LVS (Layout Versus schematic consistency check) vérifie la cohérence entre le layout et le schéma.

- LPE (Layout Parameter extractor) extrait divers paramètres du layout tels que les W et L des transistors.

- LVL (Layout Versus Layout) vérifie que deux layouts remplissent bien les mêmes fonctions.

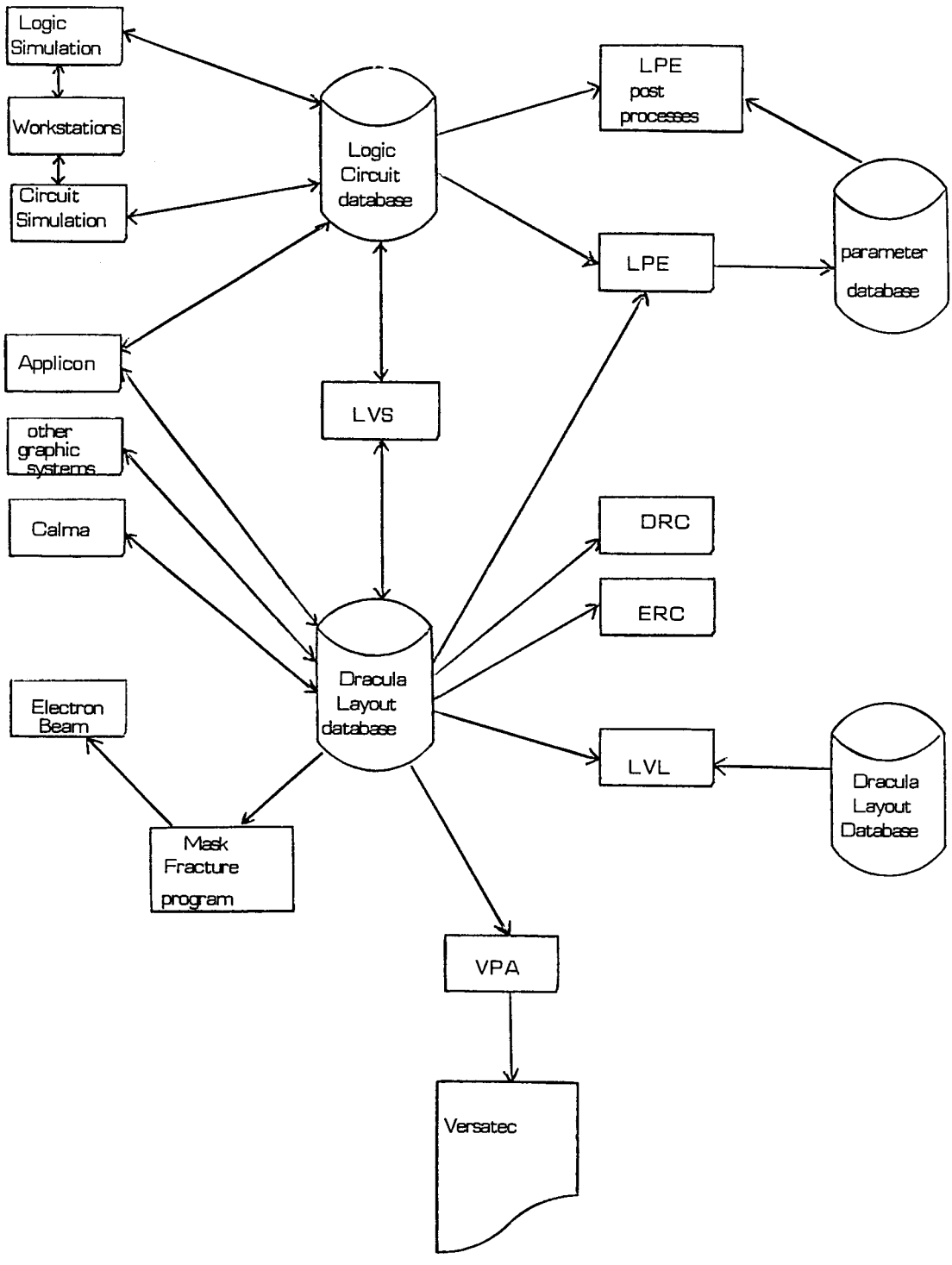
- VPA (Versatec Plot Accelerator) accélère la sortie de schémas sur les traceurs VERSATEC ou compatibles.

- DRC (Design Rule Checking) vérifie les règles de garde.

- ERC (Electrical Rule Checking) vérifie quelques règles électriques telles que la détection des court-circuits, des circuits ouverts ou des noeuds flottants.

Le schéma général de DRACULA est le suivant :

Figure 91 :


Logic Simulation
Workstations
Circuit Simulation
Logic Circuit database
LPE post processes
LPE
parameter database
Applicon
LVS
other graphic systems
Calma
DRC
ERC
Dracula Layout database
Electron Beam
LVL
Dracula Layout Database
Mask Fracture program
VPA
Versatec

Au niveau du layout, DRACULA représente les masques comme un ensemble de trapèzes de taille et forme quelconques.

a- La vérification des règles de garde

Voici quelques fonctions disponibles pour la vérification des règles de garde. Cette liste n'est pas exhaustive, mais permet de se faire une idée sur le fonctionnement général du produit.

- **CONNECT** masque 1 masque 2 **BY** masque 3

Cette fonction permet d'établir la correspondance des équipotentielles de deux niveaux différents connectés par un troisième.

- AND, OR, NOT et MINUS sont les opérateurs logiques.

- **SELECT** masque 1 **RELATION** masque 2 masque généré.

Cette fonction permet de générer un masque constitué d'éléments des deux précédents. La relation peut être : INSIDE, OUTSIDE, CUT, TOUCH (impose OUTSIDE), ENCLOSE et HOLE (génère les éléments du niveau 1 bouchant parfaitement les "trous" du niveau 2).

- SIZE permet de dilater ou contracter un masque. Dans le cas de la contraction, une option permet de maintenir ou non la tangence des différents éléments du masque.

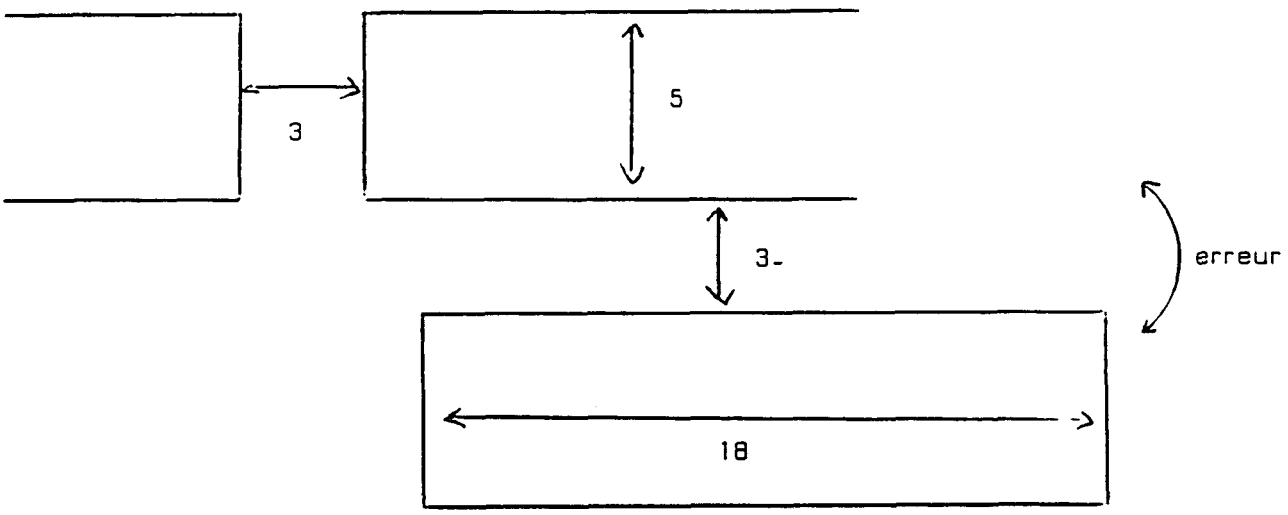
- **ENCLOSURE** masque 1 masque 2 **LT/LE/RANGE** n1(n2). Cette fonction effectue la vérification des gardes de recouvrement. On remarque qu'il est possible de définir des recouvrements maximum, minimum ou les deux. Une option permet, sur demande, de générer les polygones manquants pour que la règle soit respectée.

- **EXT** masque **LT/LE/RANGE** n1. Cette fonction effectue la vérification de garde extérieure.

- **WIDTH** masque **LT/LE/RANGE** n1. Cette fonction effectue la vérification de garde intérieure.

- **LENGTH GT/GE/LT/LE** n1. Cet opérateur est utilisé en seconde partie d'une vérification. Par exemple, EXT métal LT 4 & LENGTH métal GT 6 OUTPUT masque. Dans ce cas, on considère qu'il y a une erreur si la première règle n'est pas vérifiée dans une direction, et la seconde dans l'autre.

Figure 92 :



- **AREA RANGE** n1,n2 . Cette fonction considère comme erreur tout élément dont la surface est comprise entre n1 et n2.

- PLENGTH fonctionne comme AREA, mais pour le périmètre.

b- Définition des éléments du circuit

L'extracteur de schéma de DRACULA s'appuie sur un ensemble de fonctions spécifiques à la définition des différents composants du circuit. En voici quelques exemples.

- **ELEMENT MOS** type masque a, b, c, d permet de définir les transistors MOS constituant le circuit. Le type permet de différencier les différentes catégories de transistors. Les niveaux sont définis comme suit :

  - a : canal (intersection polysilicium-diffusion)
  - b : grille
  - c : diffusion du canal
  - d : substrat

- **ELEMENT PAD** masque a, b indique les points de contacts externes du circuit. Les niveaux sont définis comme suit :

  - a : ouverture
  - b : masque conducteur du pad au circuit

- **ELEMENT CAP** type masque a, b permet la définition des capacités du circuit. Le type permet de différencier les catégories de capacités, qui peuvent être définies entre deux noeuds de deux masques différents, ou entre deux noeuds d'un même masque.

- La définition de la valeur de ces capacités, se fait par l'intermédiaire de la fonction ATTRIBUTE CAP type val a, val b. Les valeurs sont définies comme suit :

  - val a : capacité par unité de surface
  - val b : capacité par unité de périmètre

Il existe d'autres fonctions pour définir les éléments restant, tels que les diodes ou les résistances.

c- Les contrôles

De nombreuses fonctions de contrôle du layout existent. Nous n'en ferons pas une liste exhaustive, mais une rapide présentation des opérateurs les plus représentatifs.

- PATHCHK est spécifique au MOS. Ces transistors doivent avoir, par un chemin quelconque, un noeud relié à la masse, et l'autre à l'alimentation. Cette fonction recherche ces connexions. Il est aussi possible, pour cet opérateur, de considérer que les noeuds flottants peuvent remplacer la masse ou l'alimentation, ce qui autorise la vérification de tout type de schéma.

- LVSCHK permet de vérifier la cohérence entre le schéma et le layout du circuit. Pour cet opérateur, on précise d'éventuelles tolérances sur les valeurs des paramètres W et L des transistors, ou sur le type d'élément à prendre en compte.

2) UDRC DE NCA

UDRC est un produit moins complet que DRACULA. Il n'offre que des possibilités de vérification de règles de garde (DRC) et de vérifications électriques (ERC).

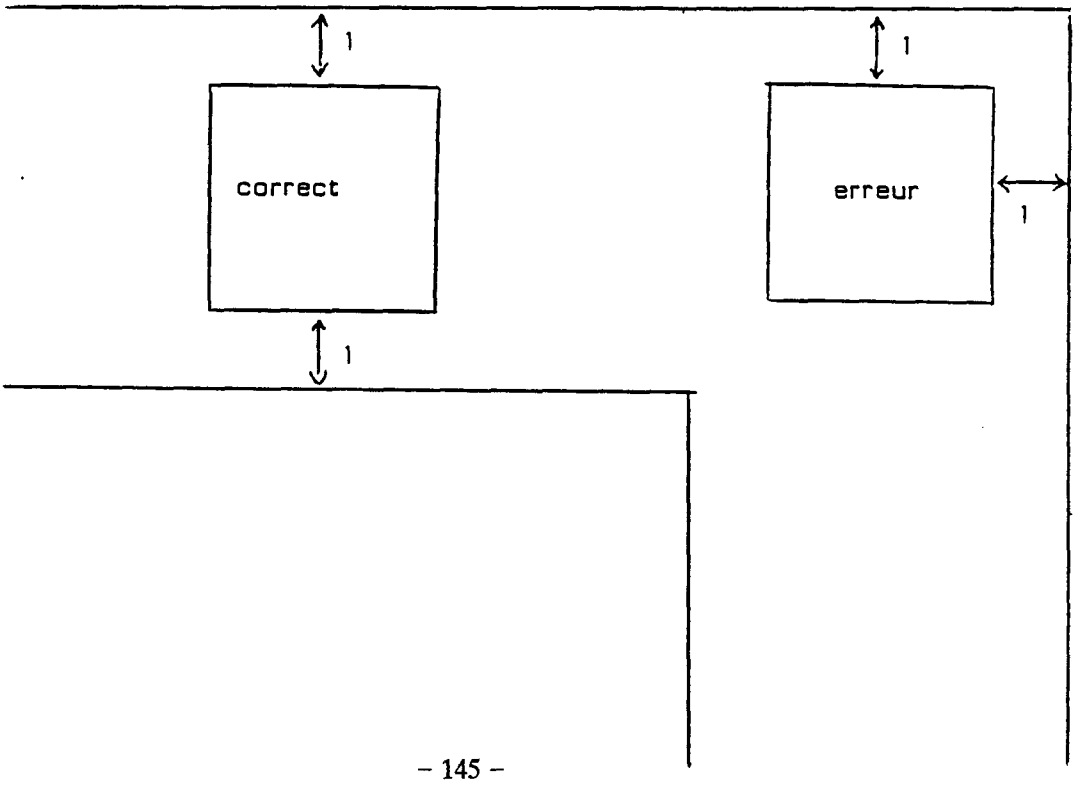
Au niveau du layout, UDRC représente les masques comme un ensemble de segments d'angle et de longueur quelconques.

a- La vérification des règles de garde

Les fonctions de UDRC et celle de DRACULA sont identiques dans leur dénomination et leur fonction. La différence réside dans le fait que UDRC autorise, pour chaque fonction, de donner des valeurs différentes suivant l'orientation ou la direction des lignes. Ainsi, il est possible de préciser des valeurs pour des lignes parallèles ou perpendiculaires à la connexion en cours de traitement. De la même façon, l'utilisateur peut spécifier des règles différentes pour des éléments appartenant à une même équipotentielle, ou à des équipotentielles différentes.

La combinaison de ces différentes possibilités permet de détecter des erreurs complexes telles que :

Figure 93 :



Ce type de vérification, de plus en plus fréquent avec les nouvelles technologies, n'est pas vérifiable par un produit tel que DRACULA.

UDRC permet un contrôle complet lors d'une vérification hiérarchique. Il autorise l'utilisateur à vérifier ou non des cellules, à les reporter ou non au niveau supérieur et permet de "trouer" ou non les différents blocs constituant le circuit.

b- Vérification des règles électriques

Pour pouvoir effectuer cette vérification, l'utilisateur doit fournir un certain nombre de renseignements tels que :

- Les caractéristiques électriques des plots d'entrées.
- Les éléments constituant le circuit (transistors, diodes, résistances, capacités...). Les fonctions de définition de ces entités ont une syntaxe très proche de celle des opérateurs de DRACULA.

Partant de là, UDRC détecte :

- Les court-circuits entre la masse et l'alimentation

- Les éléments ayant une ou plusieurs extrémités non connectées

- Les noeuds électriques non connectés à un élément

- les plots d'entrée non utilisés

- Les "collages" à 0 ou 1

- Les parties du circuit non utilisées sur un plan électrique

Cet ensemble de vérifications permet d'éviter de nombreuses erreurs de connexions sans avoir recours à une extraction suivie d'une comparaison.

CONCLUSION

Le travail présenté dans ce document porte sur la réalisation de trois logiciels dans le domaine de la conception assistée par ordinateur de circuits intégrés.

L'éditeur graphique généralisé permettant la conception de circuits imprimés, de circuits intégrés numériques et de circuits intégrés analogiques.

Un vérificateur de garde autorisant la vérification de schémas constitués de segments à angle quelconque.

Un extracteur de schémas permettant une extraction fine en vue d'une simulation précise des chemins critiques. Cet ensemble de logiciels, construit autour d'une base de données unique, et une optimisation des algorithmes et structures de données permet de porter nos produits sur des équipements de type micro-ordinateurs.

Nous avons réalisé un ensemble de produit cohérent et homogène. Ils permettent la réalisation et vérification complète des circuits intégrés numériques et analogiques.

Le concepteur peut aussi faire de la schématique pour circuits imprimés, mais sans vérification. Des tests ont prouvé qu'il était possible de concevoir des circuits LSI sur des stations de travail de type micro-ordinateurs.

Nos produits autorisent donc la conception de circuits mixtes (analogiques et digitaux). La limitation essentielle de notre solution se situe sur le manque d'aide à la conception. Il serait intéressant, pour les parties numériques, d'offrir des aspects de compilation de silicium permettant un gain de temps appréciable.

L'évolution des matériels fait que la taille mémoire et la puissance des machines deviennent des facteurs de moins en moins prépondérant. Nos développements resteront valables, mais des efforts importants pourront être fait dans le domaine de l'ergonomie et surtout, de la précision des modèles utilisés. Par contre, de part l'utilisation de solutions normalisées, l'hétérogénéïté des matériels ne devrait pas poser de problèmes insurmontables.

RESUME

Le travail présenté dans ce document porte sur la réalisation de trois logiciels dans le domaine de la conception assistée par ordinateur de circuits intégrés.

L'éditeur graphique généralisé permettant la conception de circuits imprimés, de circuits intégrés numériques et de circuits intégrés analogiques.

Un vérificateur de garde autorisant la vérification de schémas constitués de segments à angle quelconque.

Un extracteur de schémas permettant une extraction fine en vue d'une simulation précise des chemins critiques. Cet ensemble de logiciels, contruit autour d'une base de données unique, et une optimisation des algorithmes et structures de données permet de porter nos produits sur des équipements de type micro-ordinateurs.

Mots clefs

Conception Assistée par Ordinateur
Micro-électronique
Noyau de Gestion Graphique